Univerza v Mariboru
Fakulteta za elektrotehniko,
računalništvo in informatiko

Tilen Hosnar

UPORABA EVOLUCIJSKIH ALGORITMOV ZA POSTAVITEV STOLPOV V RAČUNALNIŠKI IGRI OBRANI OZEMLJE

Diplomsko delo

Maribor, september 2013

UPORABA EVOLUCIJSKIH ALGORITMOV ZA POSTAVITEV STOLPOV V RAČUNALNIŠKI IGRI OBRANI OZEMLJE

Diplomsko delo

Študent:	Tilen Hosnar
Študijski program:	Visokošolski študijski program
	Računalništvo in informacijske tehnologije
Mentor:	Dr. Matej Črepinšek
Lektorica:	Nastja Stropnik Naveršnik, univ. dipl. slovenist

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

Številka: E1041143
Datum in kraj: 08. 05. 2013, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 46/2012)
izdajam

SKLEP O DIPLOMSKEM DELU

1. **Tilnu Hosnarju,** študentu visokošolskega strokovnega študijskega programa RAČUNALNIŠTVO IN INFORMACIJSKE TEHNOLOGIJE, se dovoljuje izdelati diplomsko delo pri predmetu Uvod v evolucijske algoritme.

2. **MENTOR:** **doc. dr. Matej Črepinšek**

3. **Naslov diplomskega dela:**
   UPORABA EVOLUCIJSKIH ALGORITMOV ZA POSTAVITEV STOLPOV V RAČUNALNIŠKI IGRI "OBRANI OZEMLJE"

4. **Naslov diplomskega dela v angleškem jeziku:**
   OPTIMIZING TOWERS POSITIONS USING EVOLUTIONARY ALGORITHMS IN TOWER DEFENSE COMPUTER GAME

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela" in ga oddati v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije do 30. 09. 2013 v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

Dekan:
red. prof. dr. Borut Žalik

UNIVERZA V MARIBORU FAKULTETA ZA ELEKTROTEHNIKO RAČUNALNIŠTVO IN INFORMATIKO

Obvestiti:
- kandidata,
- mentorja,
- odložiti v arhiv.

Zahvala

Zahvaljujem se mentorju, dr. Mateju Črepinšku, za pomoč in vodenje pri opravljanju diplomske naloge.

Posebej se zahvaljujem svoji družini in prijateljem za vso podporo in spodbujanje.

Uporaba evolucijskih algoritmov za postavitev stolpov v računalniški igri Obrani ozemlje

Ključne besede: evolucijski algoritem, računalniška igra, optimizacija, Obrani ozemlje, postavitve stolpov.

UDK: 004.421:004.5(043.2)

Povzetek

Glavna naloga igralca pri računalniški igri Obrani ozemlje je poiskati dobro postavitev, ki bo uničila vse ali večino napadalnih enot. Za iskanje dobre postavitve lahko posegamo po raznih algoritmih, ki igranje olajšajo in naredijo igro še bolj zanimivo. Med te algoritme spadajo tudi evolucijski algoritmi, ki so se v preteklosti izkazali za zelo uspešne na področju reševanja zapletenih optimizacijskih problemov.

Cilj diplomske naloge je izvesti evolucijski algoritem, ki bo optimiziral postavitve stolpov pri igri Obrani ozemlje, sočasno pa implementirati inteligentnega pomočnika, ki bo igralcu s pomočjo algoritma prikazoval namige za dobre postavitve stolpov.

Optimizing towers positions using evolutionary algorithms in Tower defense computer game

Key words: evolutionary algorithm, computer game, optimization, Tower defense, towers positions.

UDK: 004.421:004.5(043.2)

Abstract

The main task of a player in the game Tower defense is to find a good layout of the towers that will destroy all or at least most of the offensive units. A good formation can be achieved using a variety of algorithms, which facilitate the game and make it even more interesting. Among these algorithms there are also the evolutionary algorithms, which have proved in the past to be very successful in solving complex optimization problems.

The objective of the diploma thesis is to implement an evolutionary algorithm, which would optimize the layout of the towers in the game Tower defense and at the same time implement also an intelligent assistant that would, with the help of the algorithm, show to the player good hints for towers placement.

KAZALO VSEBIN

KAZALO SLIK

KAZALO TABEL

1 UVOD

Evolucijski algoritmi so računalniški postopki za iskanje dobrih rešitev s posnemanjem biološke evolucije. Tekom razvoja so se spreminjali glede na potrebe reševanja zahtevnejših problemov na področju robotike, kriptiranja, navigacije, računalniških iger in drugje.

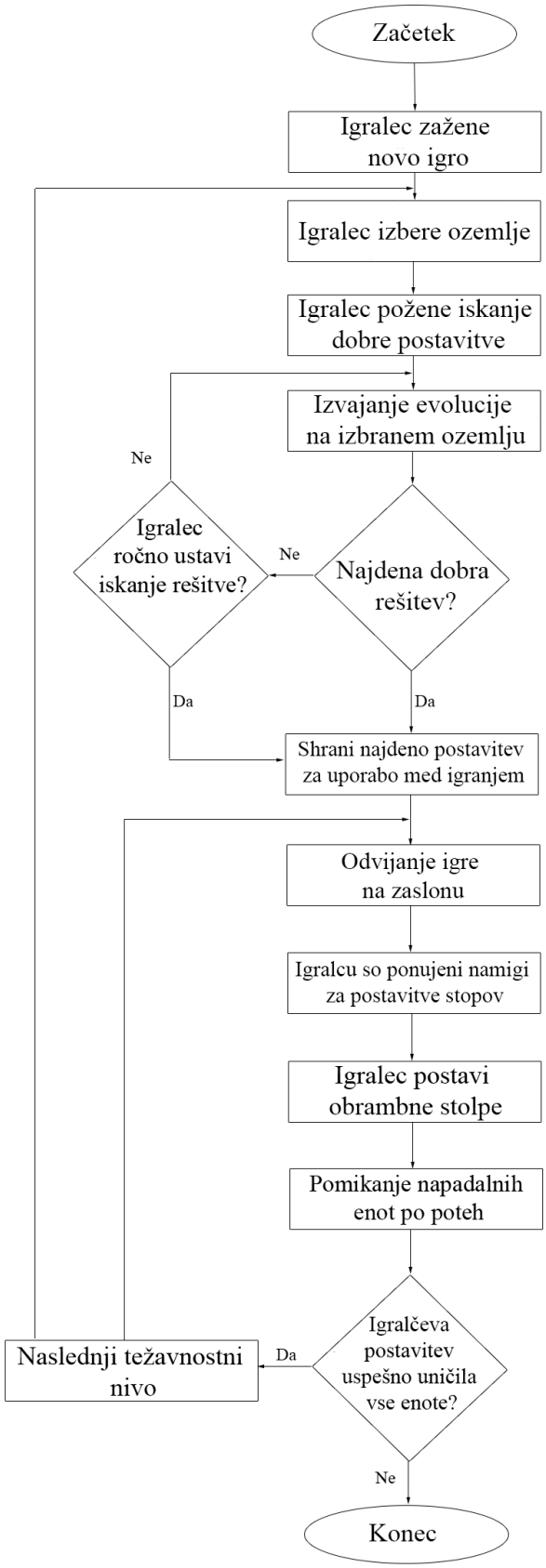
Računalniške igre so računalniški programi, ki delujejo na osnovi hitro animirane grafike na zaslonu. Namenjene so razvedrilu in zabavi uporabnikov. Spadajo med eno izmed najbolj rastočih področij v računalniškem svetu.

Cilj in namen diplomske naloge je izvesti evolucijski algoritem, ki bo optimiziral postavitve stolpov pri igri Obrani ozemlje, pri tem pa še natančneje spoznati programsko okolje XNA Game Studio in samo delovanje, prednosti evolucijskih algoritmov za izboljševanje iger in uporabnost na drugih področjih. V delu teoretično opišemo igro Obrani ozemlje in uporabo evolucijskega algoritma. Prav tako je predstavljena uporaba programskega okolja XNA Visual Studio in preostalih temeljev tega diplomskega dela. V praktičnem delu je predstavljeno reševanje problema postavitve stolpov v igri Obrani ozemlje s pomočjo evolucijskih algoritmov in uporabljene metode.

Na sliki spodaj lahko vidimo diagram poteka, ki nam prikazuje delovanje igre (Slika 1.1).
V prvem koraku si igralec izbere ozemlje, za katerega nato algoritem poišče dobro postavitev. Nato v drugem koraku s pomočjo najdene rešitve program prikaže igralcu namige za postavitve stolpov na dobre pozicije. Ko je igralec postavil vse stolpe, se igra prične. Če igralcu uspe priti do naslednjega nivoja, sta možni dve spremembi težavnosti in sicer:

- spreminjanje zmožnosti napadalnih enot ali
- spreminjanje ozemlja.

V primeru spremembe ozemlja se mora ponovno izvesti evolucija.



Slika 1.1: Diagram poteka delovanja igre.

2 RAČUNALNIŠKE IGRE

Računalniške igre, kot programska koda, so namenjene zabavi in krajšanju prostega časa. [10] Gre za elektronske igre, pri kateri uporabnik z vmesnikom upravlja računalnik, na katerem teče igra, ta pa mu prikaže odziv v video obliki. Danes igre delujejo na široki paleti naprav, ki jih s skupnim izrazom imenujemo »platforme«. To so računalniki, igralne konzole, tablice in druge. V to skupino spadajo naprave od velikih računalnikov do majhnih prenosnih naprav. Razvoj iger je v zadnjem desetletju zelo napredoval in danes si že težko predstavljamo računalnik, ki ne bi imel naložene kakšne računalniške igre. [12]

2.1 Igra Obrani ozemlje

Igra Obrani ozemlje je mešanica strategije in akcijske igre. Prototip igre, ki je bil razvit leta 1990, je imenovan Rampart. Razvilo ga je podjetje Atari Games, ki se ukvarja z razvojem arkadnih iger.

Značilnost igre Obrani ozemlje predstavlja prostor, po katerem potekajo prehodne poti. Te poti so namenjene mobilnim napadalnim enotam, katerih cilj je prehod od začetka do konca. Slednje bi jim onemogočilo končno zavzetje ozemlja. Za preprečitev njihovega cilja postavljamo različne statične obrambne stolpe. Slednji lahko imajo različen razpon in moč delovanja oziroma napada. Njihov cilj je uničevati napadalne enote in s tem varovanje ozemlja, ki ostane v lasti igralca do izgube življenj, katerih število je omejeno. To posledično vpliva na količino pridobljenega navideznega denarja, s katerim lahko razpolagamo tekom igranja (dodatni stolpi, nadgradnje, bonusi). Število uničenih napadalnih enot je odvisno od količine in postavitve stolpov, njihove moči ter razpona delovanja. [11]

3 EVOLUCIJSKI ALGORITMI

Algoritem je ime za vsak dobro definirani računalniški proces, ki kot vhod sprejme vrednosti, iz katerih nato proizvede nove izhodne vrednosti. Te vrednosti predstavljajo rešitev problema. Gre torej za zaporedje računalniških korakov, ki vhodne vrednosti preoblikujejo v izhodne vrednosti. Prav tako lahko na algoritem gledamo kot na orodje za reševanje specifičnih računalniških problemov.

Dandanes poznamo mnogo različnih algoritmov od preprostih, kot je razvrščanje števil po velikosti, pa vse do zahtevnejših, med katere štejemo predvsem optimizacijske algoritme. [3]

Glavne značilnosti algoritma so:

- ima podatke,
- vrne rezultat,
- je natančno določen,
- se vedno konča,
- mogoče ga je opraviti. [1]

Današnje življenje je rezultat procesa, katerega začetek sega milijone let nazaj. Imenujemo ga evolucija. Zaradi neverjetnih sprememb, ki so se zgodile s pomočjo evolucije, znanstveniki že leta želijo posnemati procese evolucije, vendar jim to ne uspeva, tako kot bi želeli. Problem, s katerim se srečujejo, je, da s svojimi izdelki nikakor ne znajo posnemati narave v celoti, saj ne dosegajo naravnega modela.

Posnemanje procesov narave se je razširilo tudi v računalniški svet, kjer z raznimi algoritmi poskušajo doseči oziroma posnemati delovanje evolucije in s pomočjo njih iskati rešitve za določene probleme. [8]

3.1 Splošno o evolucijskih algoritmih

Evolucijski algoritem je skupni izraz za navodila reševanja problemov z računalniško pomočjo po vzoru delovanja biološke evolucije. Za te algoritme je značilno, da simulirajo mehanizme evolucije, kot so selekcija, križanje in mutacija.

Uporabni so predvsem za reševanje zahtevnih optimizacijskih problemov. Po njih posegamo v primerih, kjer optimalne rešitve ne poznamo in nam je dovolj približna, skoraj optimalna rešitev. Veliko uspešnost kažejo na področjih, kjer uporabljamo dinamične sisteme in kjer so relacije med spremenljivkami neznane ali le delno znane. [9]

Ti algoritmi so bili do sedaj predvsem uspešni na naslednjih področjih:

- planiranje,
- optimizacijski problemi,
- avtomatsko učenje,
- podatkovno rudarjenje,
- planiranje in nadzor,
- načrtovanje,
- simulacija in identifikacija,
- krmiljenje,
- klasifikacija ... [2]

3.2 Delovanje evolucijskih algoritmov

V zadnjih nekaj desetletjih so se algoritmi, ki posnemajo naravno evolucijo, izkazali za zelo robustne, kljub temu da uporabljamo le posplošeno kopijo naravnega modela. Celotno skupino teh algoritmov imenujemo evolucijski algoritmi. Te pa delimo naprej na posamezne algoritme, ki se v današnjem času pogosto uporabljajo za iskanje rešitev pri različnih zapletenih problemih. [8]

Vsak algoritem iz skupine evolucijskih algoritmov deluje s pomočjo učenja znotraj populacije. Populacija je sestavljena iz določenega števila posameznikov, ki jih imenujemo kromosomi. V večini primerov se začetna populacija ustvari naključno.

Kromosomi vsebujejo določeno število genov, s katerimi manipulirata genetska operatorja, to sta križanje in mutacija, ter posledično spreminjata sestavo kromosomov. Pogosto proces deluje tako, da selekcija izbere kromosoma v populaciji, katera nato podvrže procesoma križanja in mutacije.

Iz okolja dobimo povratno informacijo, ki pove, kakšna je kakovost posameznih kromosomov. Ta informacija se pogosto uporablja v procesu selekcije in sicer zato, da izberemo dva najboljša kromosoma, s pomočjo katerih nato pridobimo dobre potomce.

Če gledamo na ta proces v celoti, lahko opazimo, da se kakovost populacije skozi generacije zvišuje in nas pripelje do dobre rešitve, včasih tudi do optimalne rešitve. [5]

Evolucijski algoritem lahko splošno v psevdokodi zapišemo na naslednji način:

ZAČETEK
    Naključno ustvari začetno populacijo.
    Oceni posameznike začetne populacija.
    PONAVLJAJ, DOKLER zaustavitveni kriterij ni izpolnjen.
        Križanje.
        Mutacija.
        Ocenitev.
        Selekcija.
    KONEC PONAVLJANJA
KONEC [5] [8]

3.3 Ocenitvena funkcija

Za ocenitev rešitve moramo definirati cenitveno funkcijo. Ta funkcija je v algoritmu uporabljena pri selekciji, katera s pomočjo ocene posameznih kromosomov izbere najboljše med njimi in posledično izboljšuje populacijo.

Ocenitveno funkcijo oblikujemo glede na optimizacijski problem, za katerega iščemo rešitev. V večini primerov je ocena predstavljena s številom. [4]

3.4 Genetski operatorji

Evolucijske algoritme sestavljajo trije genetski operatorji: selekcija, križanje in mutacija. Genetski operatorji omogočajo nastajanje novih rešitev na podlagi starih rešitev. Pravimo, da tvorijo naslednike. Najpogosteje se uporabljata križanje in mutacija. [8]

3.4.1 Selekcija

Selekcija predstavlja kriterij, s pomočjo katerega izbiramo kromosome, ki jih prenesemo v naslednjo generacijo. Eden od načinov izbire kromosomov za prenos v naslednjo generacijo je preko njihove ocene, katera je odvisna od tega, kako dobre rezultate so posamezni kromosomi dosegli. Ker želimo rešitve iz generacije v generacijo izboljševati, je pomembno, da v naslednjo generacijo postavimo predvsem dobre kromosome. Poznamo več različnih načinov selekcije. Nekaj najpogosteje uporabljenih:

- naključna selekcija,
- turnirska selekcija,
- selekcija po rangu,
- elitizem.

Naključna selekcija izbira kromosome, ki jih bo prenesla v naslednjo generacijo z naključno izbiro, kar pomeni, da imajo vsi kromosomi enako verjetnost izbire.

Turnirska selekcija izbira naslednike tako, da najprej izbere naključno izbrano število kromosomov iz množice kromosomov, nato pa med izbranimi kromosomi določi najboljšega posameznika in ga postavi v naslednjo generacijo.

Selekcija po rangu deluje tako, da najprej kromosome uredi po njihovi oceni, nato pa izbere starša glede na zaporedno mesto v urejenem seznamu in ne glede na oceno samo. Izbira zaporednega mesta je naključna ali pa je izbrana po v naprej določenih kriterijih.

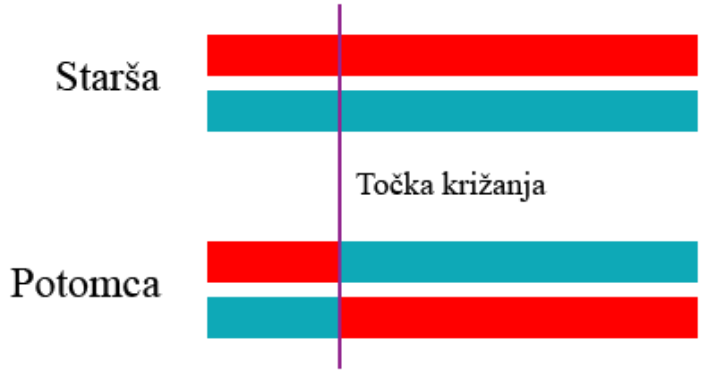
Elitizem je način izbiranja naslednikov, pri katerem se lahko rezultati le izboljšujejo ali pa ostajajo enaki. Deluje tako, da določeno število najbolje ocenjenih kromosomov vedno prenese v naslednjo generacijo, kar pomeni, da kakovost rešitve ne izgubimo. Običajno je delež teh izbranih kromosomov odvisen od problema, ki ga rešujemo. V praksi število elite redko presega 10 % celotne populacije. [6]

3.4.2 Križanje

Križanje dveh ali več staršev, ki smo jih izbrali s pomočjo selekcije, določa odstotek križanja, katerega označujemo z oznako p_c. Križanje omogoča z izmenjavo posameznih lastnosti posameznikov ustvarjanje novih posameznikov. Dele genskega materiala, ki se bodo pri križanju izmenjali, lahko določimo na različne načine. V nadaljevanju bomo opisali dva enostavna načina križanja:

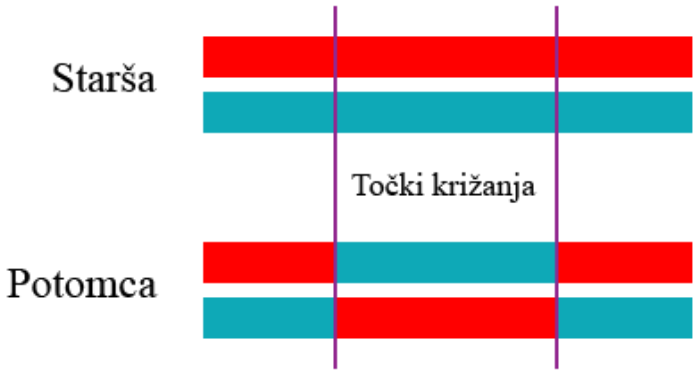
- enotočkovno križanje,
- dvotočkovno križanje.

Enotočkovno križanje deluje tako, da naključno izberemo točko v kromosomu in od te točke naprej izmenjamo genski material. Primer takšnega križanja lahko vidimo spodaj (Slika 3.1).



Slika 3.1: Enotočkovno križanje.

Dvotočkovno križanje deluje tako, da naključno izberemo dve točki v kromosomu in izmenjamo genski material med tema dvema točkama. Primer takšnega križanja lahko vidimo spodaj (Slika 3.2). [6]

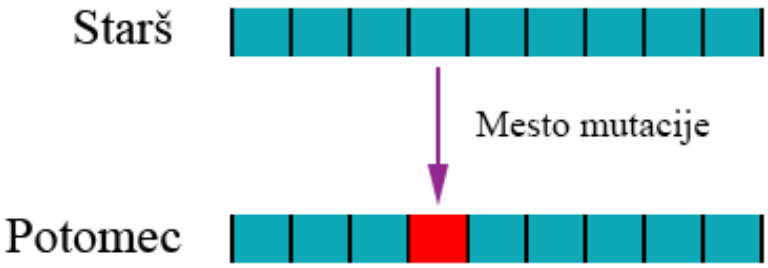


Slika 3.2: Dvotočkovno križanje.

3.4.3 Mutacija

Mutacija deluje na principu naključnega spreminjanja genskega materiala v kromosomih. Da pride do mutacije, je odvisno od odstotka mutacije, ki ga označimo z oznako p_m. Ta odstotek je običajno zelo nizek, ampak je vseeno mutacija zelo pomembna za evolucijske algoritme, saj z njeno pomočjo preprečimo enoličnost populacije.

Za vsak gen znotraj posameznika se preverja verjetnost mutacije. Če pride do mutacije, se ta gen zamenja z naključnim, kar lahko vidimo na primeru spodaj (Slika **3.3**). V primeru, da do mutacije ne pride, potomca brez sprememb prenesemo v naslednjo generacijo. [6]



Slika 3.3: Mutacija.

3.5 Kriterij zaustavljanja algoritma

Evolucija v naravi se nikoli ne zaustavi, ampak v svetu računalništva moramo nekako določiti konec izvajanja algoritma.

Evolucijski algoritem kot slučajnostni algoritem daje možnost, da ne najde optimalne rešitve. V ta namen nam kriteriji zaustavljanja omogočajo iskanje dobre rešitve.

Definiranje kriterija zaustavljanja velja za eno težjih nalog pri evolucijskih algoritmih, saj ne moramo vedeti, kako bodo ti delovali na določenih problemih. Res je, da se kromosomi izboljšujejo skozi generacije, ampak ne vemo, kdaj so dovolj dobri, da izvajanje algoritma zaustavimo. Posledično je prišlo do različnih idej in načinov, kako lahko definiramo kriterij zaustavljanja. [14]

V nadaljevanju je naštetih nekaj kriterijev, ki so najpogosteje uporabljeni (uporabimo lahko tudi kombinacijo več pogojev hkrati):

- časovna omejitev izvajanja algoritma,
- določeno število generacij,
- največje število ocenjenih posameznikov,

- ponavljanje rešitev v populaciji,
- ponavljanje rešitev posameznikov,
- ročno zaustavljanje. [4] [14]

3.6 Delitev evolucijskih algoritmov

Evolucijske algoritme lahko delimo na evolucijske strategije, genetske algoritme in evolucijsko programiranje. Kasneje se še iz genetskih algoritmov razvijejo še klasifikatorski sistemi in genetsko programiranje.

Naše diplomsko delo temelji na evolucijskemu algoritmu, ki ima zelo podoben način delovanja, kot ga ima genetski algoritem. Razlikujeta se v tem, da genetski algoritem za predstavitev populacije uporablja bitne nize, naš algoritem pa namesto teh uporablja pozicije stolpov. Natančnejši opis genetskih algoritmov in njihovega delovanje je opisan v nadaljevanju, in sicer v razdelku 3.6.1.

3.6.1 Genetski algoritmi

Genetski algoritem velja za enega najpogosteje uporabljenih algoritmov v skupini evolucijskih algoritmov. Deluje na osnovi kodiranja bitnega niza, ki vsebuje samo enke in ničle.

Prvo vprašanje pri genetskemu algoritmu je, kako nastaviti funkcijo uspešnosti. Ta določa uspešnost ohranjanja rešitve, kar pomeni, da bodo boljše rešitve imele več možnosti za ohranitev kot slabše rešitve.

Deluje na osnovi krmilnih parametrov, kateri morajo biti nastavljeni pred začetkom izvajanja. Sem spadajo parametri, velikost populacije, verjetnost križanja, verjetnost mutacije in število generacij.

V začetku izvajanja algoritem najprej ustvari populacijo in njene nize ovrednoti, tako da lahko loči med uspešnimi in manj uspešnimi nizi. Ta ovrednotenja uporablja selekcija, katera izbere nize, ki jih bo algoritem podvrgel operatorjema križanja in mutacije.

Izbira nizov, ki bodo podvrženi procesu križanja, poteka s pomočjo prednastavljenega parametra, ki ga imenujemo verjetnost križanja. Postopek izbire deluje tako, da algoritem najprej generira naključne vrednosti na intervalu od 0 do 1. Število teh naključnih generiranj je enako velikosti populacije. Če je naključno generirana vrednost manjša od parametra verjetnosti križanja, potem algoritem proces križanja izvede. V nasprotnem primeru izbrani niz ostane enak.

Za križanju je na vrsti mutacija, ki spreminja le posamezne bite v nizih. Podobno kot pri križanju, ima tudi mutacija vnaprej določen parameter, ki ga imenujemo verjetnost mutacije. Algoritem ponovno generira naključne vrednosti na intervalu od 0 do 1. Število teh naključnih generiranj je enako velikosti populacije. Če je vrednost naključnega števila manjša od parametra verjetnosti mutacije, potem se mutacija izvede. V nasprotnem primeru izbrani niz ostane enak. Sedaj so nizi pripravljeni na ponovno ovrednotenje. [9]

4 UPORABLJENO PROGRAMSKO OKOLJE

Za izdelavo projekta, na katerem temelji moje diplomsko delo, sem uporabil program XNA Game Studio.

4.1 XNA Game Studio

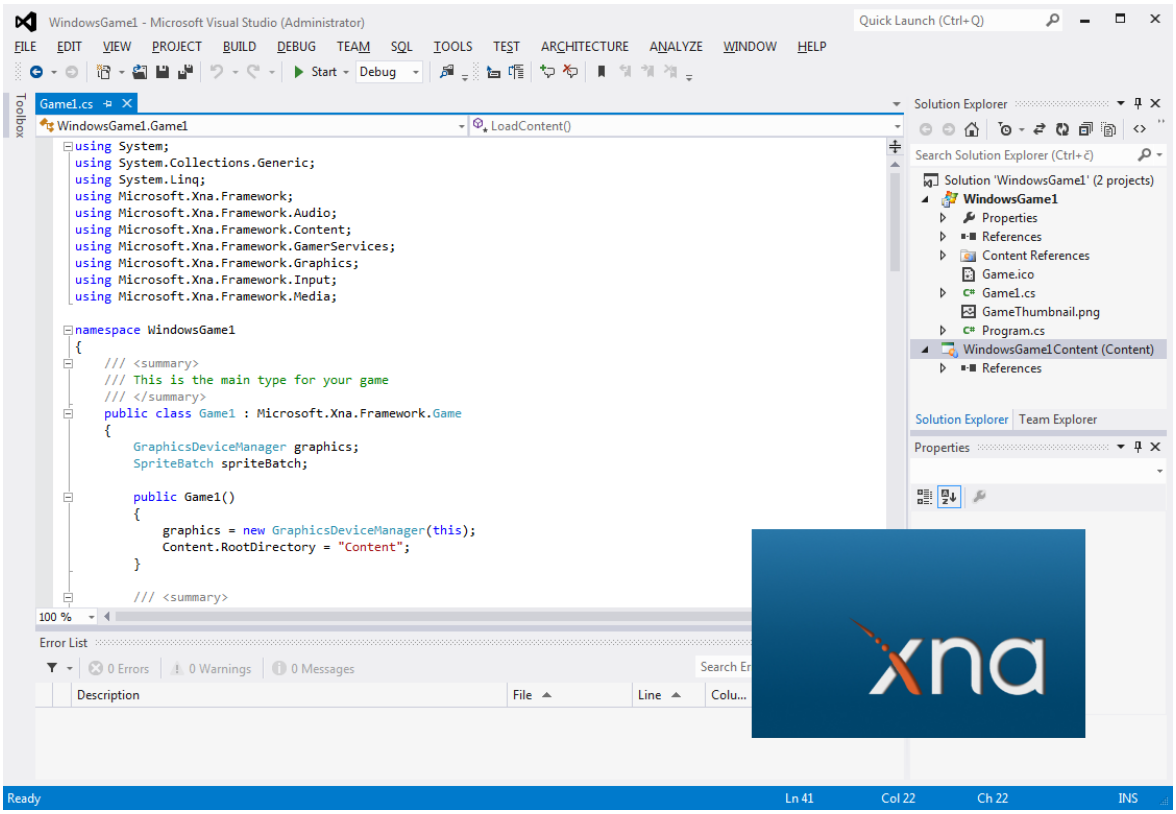
XNA Game Studio je programsko okolje, ki nam omogoča, da z uporabo Visual Studia izdelamo igre. Vsebuje XNA framework in set knjižic za razvoj iger, ki temeljijo na Microsoft .NET frameworku. V XNA lahko izdelamo igre, ki temeljijo na naslednjih platformah:

- Windows Phone,
- Xbox 360,
- Windows. [13]

Razvoj iger pogosto od razvijalcev zahteva, da izdelajo kodo, ki animira pomike in definira druge grafične aspekte. Če lahko enako kodo nato uporabijo v drugih programskih okoljih, je to za podjetja zelo koristno, saj precej zmanjša čas razvoja iger. V tem pogledu zelo pripomore XNA Game Studio, ki je zasnovan ravno na takšen način. Razvijalci se posledično lahko osredotočijo na sam razvoj igre, medtem pa zasnovane knjižice poskrbijo za manjše podrobnosti in implementacijo v druga okolja. XNA nudi podporo tako za 2D kot za 3D igre.
Poleg tipkovnice in miške za računalnik omogoča tudi uporabo Xbox kontrolorjev za testiranje iger med samim razvojem (v primeru, da izdelujemo igro za Xbox platformo).

Igre, izdelane za Xbox platformo, se ne smejo distribuirati prostovoljno, medtem ko namizne igre vsebujejo prostovoljno distribucijo licence. [7]

Uporabniški vmesnik XNA Game Studio programskega okolja je prikazan spodaj (Slika 4.1).



Slika 4.1: Programsko okolje XNA Game Studio.

5 UPORABA EVOLUCIJSKIH ALGORITMOV ZA POSTAVITEV STOLPOV V RAČUNALNIŠKI IGRI OBRANI OZEMLJE

5.1 Opredelitev problema

Cilj računalniške igre Obrani ozemlje je učinkovito in uspešno obraniti ozemlje pred napadalnimi enotami s pomočjo obrambnih stolpov. Uspešnost definira čim manjše število uporabljenih obrambnih stolpov, učinkovitost pa predstavlja hitrost obrambe in dosega cilja. Dobra postavitev je tista, s katero je mogoče uničiti vse ali večino napadalnih enot.

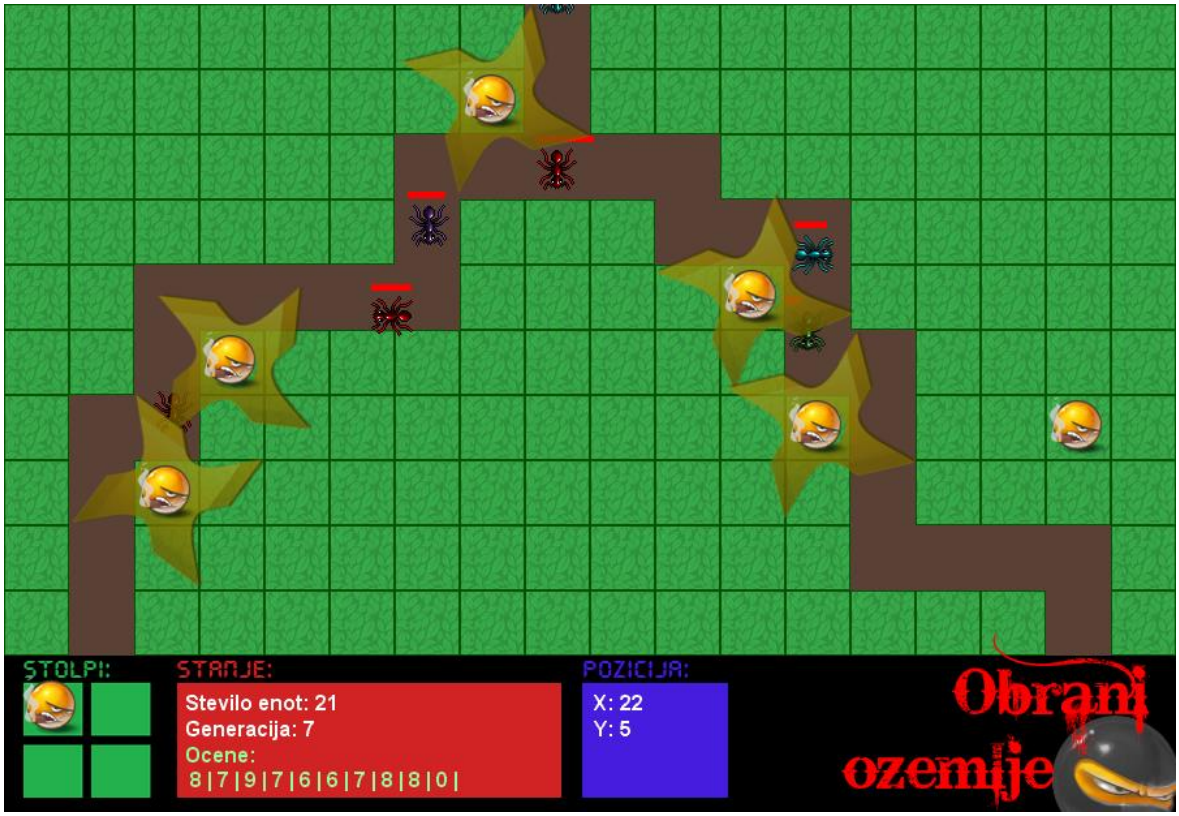
Za dosego cilja raziščemo, izdelamo in kasneje implementiramo evolucijski algoritem, ki v kratkem času poišče dobro postavitev stolpov ne glede na izbrano ozemlje. Z njegovo pomočjo nato v igro dodamo inteligentnega pomočnika, in sicer sistem za namige, ki igralcu pomaga pri dobri postavitvi. Namigi igralcu olajšajo igro, saj lahko na podlagi predlogov programa postavlja stolpe na pozicije, kjer bodo dosegali dobre rezultate. Posledično inteligentni pomočnik naredi igro še bolj zanimivo. Populacijo iskanja na izbranem ozemlju predstavlja *n* postavitev, vsaka pa vsebuje *m* stolpov.

Za izboljšanje rezultatov je potrebno izvesti večje število testiranj, pri katerih nastavljamo različne vrednosti začetnih parametrov in opazujemo rezultate. Nastavitev, ki prinaša najboljše rezultate, nato obdržimo. Prav tako za iskanje rešitev implementiramo dodatne metode, s pomočjo katerih preskočimo zamudne korake algoritma in posledično pospešimo iskanje.

Nazadnje je potrebno določiti zaustavitveni kriterij algoritma, ki bo algoritem zaustavil na določeni točki. V večini primerov ta točka predstavlja že najdeno dobro rešitev. Ta kriterij je najbolje nastaviti po večjem številu testiranj, saj takrat ugotovimo, kakšno je povprečno delovanje algoritma in čas iskanja rešitve. S pomočjo ugotovitev testiranja lažje določimo točko, na kateri se bo algoritem nehal izvajati.

Za izdelavo igre in evolucijskega algoritma bomo uporabili programsko okolje XNA Game Studio (natančneje opisan v razdelku 4.1).

Spodaj vidimo primer izvajanja algoritma in iskanja dobre postavitve (Slika 5.1). Zeleni kvadratki v naši igri predstavljajo ozemlje, kjer lahko postavljamo obrambne stolpe. Pot, po kateri se pomikajo napadalne enote, pa predstavljajo rjavi kvadratki. Za predstavitev napadalnih enot smo uporabili slike mravelj, rdeča črta nad njimi pa predstavlja količino njihovega življenja. Ta črta se zmanjšuje sorazmerno s količino življenja, ki ga obrambni stolpi vzamejo napadalni enoti. Obrambni stolpi so predstavljeni s pomočjo rumenih obrazov, ki jih lahko vidimo na sliki spodaj. Pod ozemljem pa imamo še predel, ki vsebuje informacije o izvajanju algoritma, kot so število enot, števec generacij, čas izvajanja, ocene postavitev in druge.



Slika 5.1: Primer izvajanja algoritma.

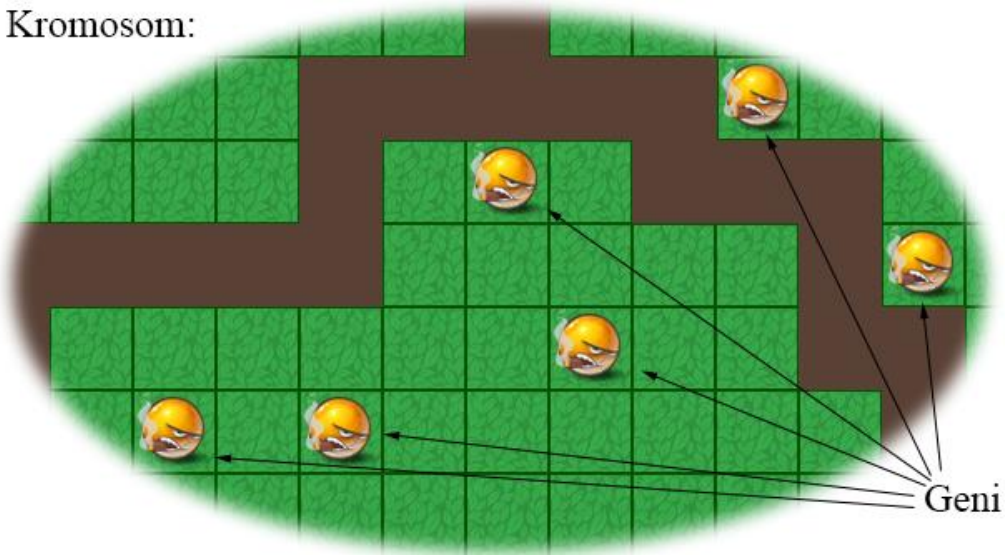
5.2 Predstavitev problema

Evolucijski algoritem rešuje probleme na osnovi populacije. Populacijo sestavljajo posamezne postavitve, ki jih imenujemo kromosomi. Ti vsebujejo določeno število obrambnih stolpov, katere imenujemo geni. Z njimi manipulirata genetska operatorja, križanje in mutacija, ter posledično spreminjata sestavo kromosoma (v našem primeru en kromosom sestavlja 6 genov). Vsak stolp – gen vsebuje lastnosti, na osnovi katerih algoritem išče dobro postavitev. Med glavne lastnosti spadata:

- pozicija stolpa,
- ocena stolpa.

Vsota ocen stolpov – genov predstavlja oceno kromosoma. Ta ocena se uporablja v procesu selekcije, saj na podlagi tega ocena izbira dobre potomce za naslednjo generacijo. Ocenjevanje kromosomov in način ocenjevanja je natančneje opisan v nadaljevanju, in sicer v razdelku 5.2.1.

Predstavitev kromosoma lahko vidimo na sliki spodaj (Slika 5.2).



Slika 5.2: Predstavitev kromosoma.

5.2.1 Ocenitvena funkcija

Ocenitvena funkcija je način ocenjevanja postavitev – kromosomov. V našem primeru smo to funkcijo definirali tako, da najprej ocenjuje posamezne stolpe – gene glede na njihove dosežke. Določanje ocene posameznega stolpa poteka tako, da vsako postavitev v populaciji program postavi na ozemlje in nato po poteh pošlje napadalne enote. Stolpom se nato ocena povečuje sorazmerno, s količino življenja, ki so ga vzamejo napadalnim enotam. Seštevek vseh ocen stolpov v posameznih postavitvah predstavlja oceno postavitve. Ker imajo napadalne enote nekje med 3000 in 6000 enot življenja, lahko posledično vsaka postavitev doseže oceno med 0 in 100000, odvisno od tega, kako dobro se je posamezna postavitev izkazala. Za lažje razumevanje in preglednost, nato program te ocene zaokroži in skrajša ter jih prikaže na lestvici od 1 do 10.

Spodaj vidimo, kako so ocene prikazane med izvajanjem algoritma (Slika 5.3). Izpis prikazuje ocene za vsako postavitev v populaciji glede na dosežke v trenutni generaciji. Na sliki lahko opazimo, da je ena od postavitev v deveti generaciji dosegla oceno 10, kar pomeni, da so obrambni stolpi v tej postavitvi uspeli uničiti vse napadalne enote. Algoritem nato na osnovi teh ocen izbira potomce, s pomočjo katerih bo gradil naslednjo generacijo.

Slika 5.3: Prikaz ocen posameznih postavitev.

5.3 Algoritem

Evolucijski algoritem deluje tako, da skozi generacije poskuša najti čim boljšo postavitev obrambnih stolpov, ki bi uničila vse ali večino napadalnih enot. Postavitve so predstavljene kot kromosomi, ki so sestavljeni iz obrabnih stolpov, ki predstavljajo gene. Za iskanje

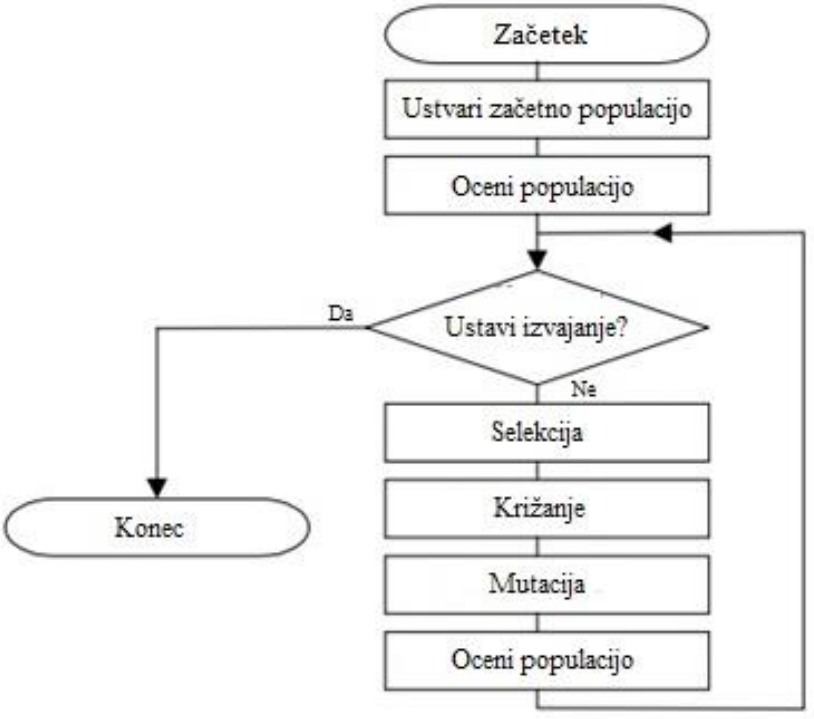
dobre rešitve algoritem uporablja genetske operatorje, in sicer selekcijo, križanje in mutacijo.

Za bolj optimalno delovanje in hitrejše iskanje rešitve smo pri mutaciji izdelali in implementirali dodatne metode (opisane v razdelku 5.6). Skozi izvajanje programa se posamezne postavitve ocenjujejo glede na število uničenih enot. Večja ocena predstavlja večje število uničenih enot.

Izvajanje programa lahko:

- zaustavi uporabnik, ko se mu postavitev oziroma ocena postavitve zdi dovolj dobra za igro,
- zaustavi program, ko ena od postavitev uniči vse napadalne enote trikrat zaporedoma, kar pomeni, da smo prišli do dobre postavitve in da smo se približali optimalni rešitvi.

Spodaj vidimo diagram poteka delovanja našega algoritma (Slika 5.4).



Slika 5.4: Diagram poteka delovanja algoritma.

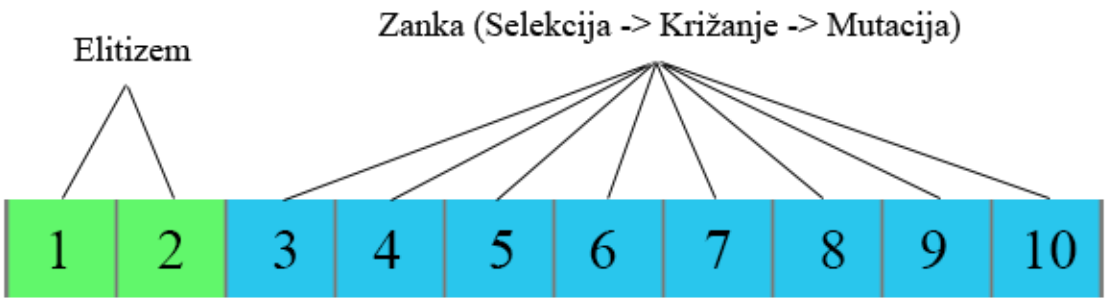
5.4 Postopek reševanja problema

Reševanja problema se začne z generiranjem naključnih postavitev. Število teh postavitev je vnaprej določeno s parametrom, ki ga imenujemo velikost populacije (pop_size). V našem primeru smo uporabili velikost populacije 10. Te postavitve predstavljajo kromosome, množica vseh kromosomov skupaj pa predstavlja začetno populacijo. V našem primeru vsaka postavitev vsebuje šest obrambnih stolpov, ki predstavljajo gene.

Program postavi prvo postavitev v populaciji na teren in pošlje napadalne enote po prehodnih poteh. Obrambi stolpi se aktivirajo, če so napadalne enote v njihovem dometu. Ob napadu se jim zvišuje ocena, ki se zvišuje sorazmerno s količino življenja vzetega napadalnim enotam. Ocene se seštevajo in prikazujejo na lestvici od 1 do 10. Postopek se ponavlja, dokler niso ocenjene vse postavitve v populaciji.

Ko so vse postavitve ocenjene, program izbere dve najbolje ocenjeni in ju prenese v naslednjo generacijo. S tem zagotovi, da naslednja generacija ne bo slabša od trenutne. Ta način izbiranja kandidatov za naslednjo generacijo imenujemo elitizem.

V naslednjem koraku se algoritem začne izvajati v zanki, ki se v našem primeru ponovi štirikrat, saj evolucijski algoritem v vsakem krogu zanke postavi dva kromosoma v naslednjo generacijo in glede na to, da smo pred začetkom zanke že zapolnili dva mesta v naslednji generaciji, imamo sedaj prostih še osem mest. Zanka se mora izvesti tolikokrat, da bo v naslednji generaciji enako število kromosomov kot v prvi. Zgoraj opisani način zapolnjevanja mest naslednje generacije je prikazan spodaj (Slika 5.5).



Slika 5.5: Način izbiranja kandidatov za naslednjo generacijo.

V zanki so sedaj na vrsti genetski operatorji. Prva med njimi je selekcija, ki v vsaki izvedbi zanke izbere dve postavitvi – starša. Ta posreduje naprej k naslednjemu operatorju. Podrobnejši opis uporabljene selekcije je prikazan v razdelku 5.7.

Naslednji genetski operator v vrsti, ki smo ga uporabili, je križanje. Križanje je proces izmenjave genskega materiala med kromosomoma, v našem primeru to pomeni izmenjava stolpov med izbranima postavitvama. Križanje se ne izvede vedno, saj je odvisno od začetnega parametra, ki ga imenujemo verjetnost križanja. Podrobnejši opis uporabljene metode križanja je prikazan v razdelku 5.5.

Zadnji genetski operator v vrsti izvajanja je mutacija. V splošnem opisu algoritma mutacija zamenja enega izmed stolpov – genov z naključno generiranim in s tem preprečuje, da bi se rešitve začele ponavljati. V našem primeru je poleg naključno generiranih genov še izdelanih in dodanih nekaj dodatnih metod mutacije, ki pomagajo algoritmu hitreje poiskati dobro rešitev. Mutacija se ne izvede vedno, saj je odvisna od začetnega parametra, ki ga imenujemo verjetnost mutacije. Ta je v večini primerov manjša kot pa verjetnost križanja. Podrobnejši opis uporabljenih metod mutacije je prikazan v razdelku 5.6 in njegovih podrazdelkih.

Ko se zanka, v kateri se izvajajo genetski operatorji, zaključi, se populacija ponovno oceni. To pomeni, da program ponovno postavi vsako postavitev v populaciji na ozemlje in jo oceni. Na podlagi teh ocen kasneje selekcija ponovno izbira potomce, ki bodo gradili novo generacijo.

Korake našega algoritma lahko predstavimo z naslednjo psevdokodo:

ZAČETEK
    Ustvari naključno množico k kromosomov.
    PONAVLJAJ NESKONČNO
        Izračunaj ocene vseh kromosomov.
        Uredi kromosome po ocenah.
        Ustvari prazno množico, ki predstavlja naslednjo generacijo.
        Shrani prvi najbolje ocenjeni kromosom.
        Shrani drugi najbolje ocenjeni kromosom.
        Dodaj prva dva najbolje ocenjena kromosoma v naslednjo generacijo.

ČE uporabnik zaustavi izvajanje algoritma, **POTEM**
VRNI prvi najbolje ocenjeni kromosom.

PONAVLJAJ, DOKLER i < velikostPopulacije/2.
S selekcijo izberi dva kromosoma-starša.
ČE naključna vrednost < verjetnost križanja, **POTEM**
križaj izbrana kromosoma,
DRUGAČE
ohrani izbrana kromosoma brez sprememb.

ČE naključna vrednost < verjetnost mutacije, **POTEM**
mutiraj izbrana kromosoma,
DRUGAČE
ohrani izbrana kromosoma brez sprememb.
Dodaj kromosoma v naslednjo generacijo.
KONEC PONAVLJANJA
KONEC PONAVLJANJA
KONEC

Postopek reševanja problema:

1. Določanje cenitvene funkcije (glej razdelek 5.2.1).
2. Izbira genetskega operatorja križanje (glej razdelek 5.5).
3. Izbira genetskega operatorja mutacija in njenih metod (glej razdelek 5.6).
4. Izbira genetskega operatorja selekcija (glej razdelek 5.7).
5. Določanje kriterija za ustavljanje (glej razdelek 5.8).
6. Določanje začetnih parametrov (glej razdelek 0).
7. Testiranje programa (glej razdelek 5.10).

5.5 Križanje

Izvajanje križanja je odvisno od nastavljenega začetnega parametra, ki ga imenujemo verjetnost križanja in ga označimo z p_c. Program v tem koraku generira naključno vrednost na lestvici od 0 do 1. Če je ta vrednost manjša od vrednosti verjetnosti križanja, se križanje izvede, v nasprotnem primeru se križanje ne izvede in kandidata nadaljujeta na naslednji korak nespremenjena.

V našem algoritmu smo izbrali enotočkovno križanje, ki deluje tako, da program najprej naključno generira vrednost na lestvici od 1 do vrednosti velikosti postavitve. Generirano vrednost imenujemo točka križanja. Ta točka predstavlja položaj v starševskima postavitvama in od tega položaja naprej se med njima izmenjajo stolpi – geni. Stolpi, ki se nahajajo pred tem položajem, ostanejo nespremenjeni.

V primeru, da je točka križanja 3, bi se križanje izvedlo tako, da bi pozicije stolpov med mestom 3 in vključno z mestom 6 izmenjali med postavitvama. Pozicije stolpov, ki se nahajajo pred 3 mestom, bi ohranili in tisti del postavitev pustili nespremenjen.

5.6 Mutacija

Izvajanje mutacije je odvisno od nastavljenega začetnega parametra, ki ga imenujemo verjetnost mutacije in ga označimo z p_m. Program v tem koraku generira naključno vrednost na lestvici od 0 do 1. Če je ta vrednost manjša od vrednosti verjetnosti mutacije, se mutacija izvede, v nasprotnem primeru se mutacija ne izvede in postavitvi nadaljujeta na naslednji korak nespremenjeni.

Program naključno izbere stolp v postavitvi – staršu in mu zamenja pozicijo. V splošni različici evolucijskega algoritma je mutacija zgolj naključna, kar pomeni, da izbranemu stolpu zamenja trenutno pozicijo z naključno generirano pozicijo. V našem algoritmu so izdelane in dodane metode, ki omogočajo hitrejše iskanje rešitve. Zaradi teh metod je korak mutacije zelo pomemben, kar je razlog za nastavljeno malo večjo vrednost parametra verjetnost mutacije, kot pa je ta običajno nastavljena v splošnih različicah algoritma.

Mutacija deluje v več korakih oziroma več različnih metodah, ki si sledijo po vrsti v naslednjem zaporedju:

1. naključna metoda,
2. metoda levo–desno,
3. metoda gor–dol.

Glavni cilj teh metod je poskrbeti za naključnost rešitve, sočasno pa omogočajo tudi iskanje pozicij stolpov ob prehodnih poteh, kjer imajo največji vpliv na napadalne enote s čimer dosegajo boljše rezultate. To omogoča hitrejše iskanje dobrih rešitev. Izbrane metode so natančneje opisane v nadaljevanju tega poglavja.

5.6.1 Naključna metoda

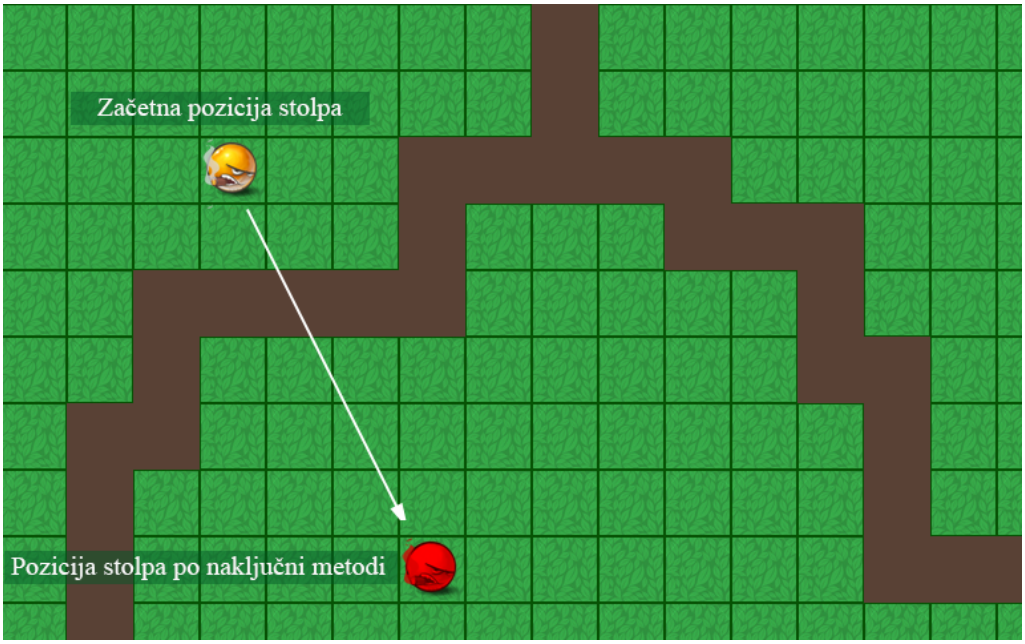
Če v algoritmu pride do mutacije, se v prvem koraku izvede naključna metoda. Namen te metode je dodajanje naključno generiranih vrednosti, kar preprečuje, da bi se rešitve začele ponavljati na določeni točki, li ne predstavlja končne rešitve**.**

Deluje tako, da generira novo naključno pozicijo, ki se nahaja znotraj meja ozemlja in nato to pozicijo nastavi mutiranemu genu – stolpu. Pri tem pa moramo paziti, da nova generirana pozicija že ni zasedena in da ne predstavlja prehodno pot, saj če se to zgodi, moramo generirati novo pozicijo. Ko se naključna metoda dokončno izvede, algoritem nato kliče naslednjo metodo mutacije in sicer metodo levo–desno.

Postopek delovanja naključne metode lahko predstavimo z naslednjo psevdokodo:

ZAČETEK
    Metoda kot parameter sprejme izbrani gen za mutiranje.
    Generiraj naključno pozicijo novega stolpa.
    PONAVLJAJ, DOKLER je pozicija zasedena.
        Ponovno generiraj naključno pozicijo novega stolpa.
    KONEC PONAVLJANJA
    Stolpu shrani novo pozicijo.
    Kliči metodo levo–desno.
KONEC

Za lažje razumevanje je spodaj prikazan potek zamenjave pozicije pri naključni metodi (Slika 5.6).



Slika 5.6: Naključna metoda.

5.6.2 Metoda levo-desno

Ta metoda poišče pozicijo zraven prehodne poti na vodoravni liniji. To stori tako, da glede na trenutno pozicijo stolpa program pregleda mesta levo in desno od stolpa ter poišče najbližjo prehodno pot.

Deluje tako, da najprej kot parameter sprejme pozicijo stolpa, ki je bila generirana z naključno metodo. Nato program glede na trenutno vrstico, v kateri se stolp nahaja, shrani v seznam vsa mesta, ki predstavljajo ozemlje in prehodno pot. V naslednjem koraku s pomočjo seznama pridobi pozicijo prehodne poti, ki se v vrstici nahaja najbližje našemu stolpu in s pomočjo te pozicije izračuna mesto, ki se nahaja zraven poti. To mesto predstavlja novo pozicijo stolpa. Na zadnje še program preveri, če je na novo najdena pozicija že zasedena. V primeru, da je zasedena, program ponovno kliče naključno metodo in se vse skupaj ponovi. V nasprotnem primeru pa trenutno mutiranemu genu – stolpu nastavimo novo pozicijo. Ko se metoda levo–desno dokončno izvede, algoritem nato kliče naslednjo metodo, in sicer metodo gor–dol.

Postopek delovanja metode levo–desno lahko predstavimo z naslednjo psevdokodo:

ZAČETEK

Metoda kot parameter sprejme spremenjeni gen iz naključne metode.

PONAVLJAJ, DOKLER pozicija trenutnega kvadrata < širina ozemlja.

V seznam shrani mesta kvadratov v trenutni vrstici, kjer se nahaja prehodna pot (med iskanjem se premikamo po en kvadrat naprej ali nazaj v vrstici).

KONEC PONAVLJANJA

ČE seznam mest prehodnih poti ni prazen, **POTEM**

izračunaj pozicijo kvadratka poti, ki se nahaja najbližje trenutni poziciji.

Pridobi pozicijo kvadratka, ki se nahaja zraven poti.

ČE je pozicija zasedena, **POTEM**

ponovno kliči naključno metodo.

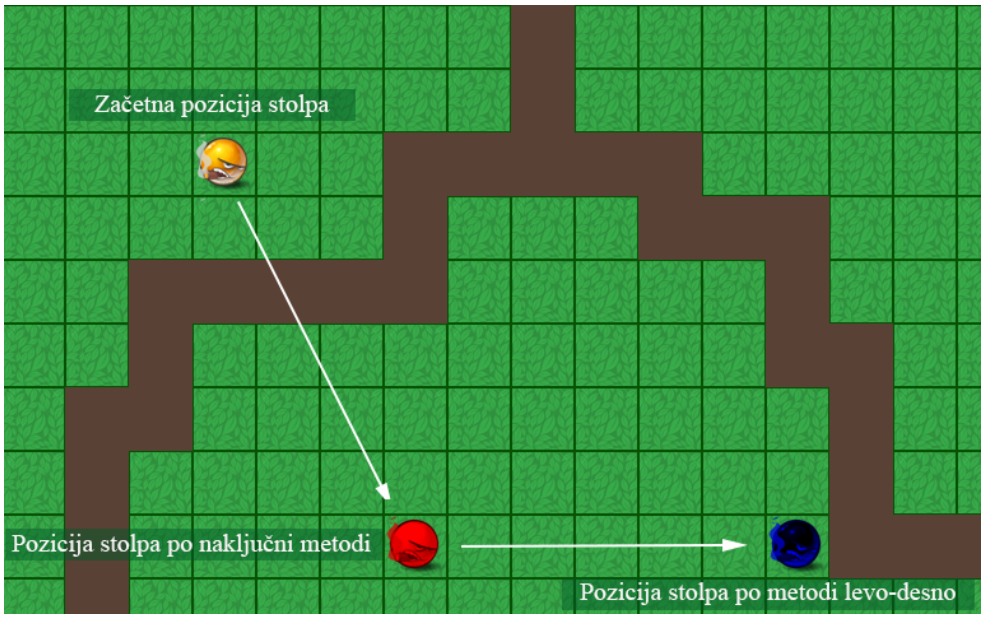
DRUGAČE

Stolpu shrani novo pridobljeno pozicijo.

Kliči metodo gor–dol.

KONEC

Za lažje razumevanje je spodaj prikazano, kako poteka zamenjava pozicije pri metodi levo–desno (Slika 5.7).



Slika 5.7: Metoda levo-desno.

5.6.3 Metoda gor–dol

Metoda gor–dol poišče pozicijo zraven prehodne poti na navpični liniji. To stori tako, da glede na trenutno pozicijo stolpa program pregleda mesta nad in pod stolpom ter poišče najbližjo prehodno pot. Metoda gor–dol postavi stolp v kot, kjer lahko ima največji vpliv na napadalne enote, zato se posledično ta metode ne izvede vedno, saj je stolp v nekaterih primerih postavljen tako, da v bližini ni kota, v katerega bi se lahko postavil. V tem primeru se izvedeta zgolj prvi dve metodi mutacije, in sicer naključna metoda in metoda levo–desno.

Metoda gor–dol kot parameter sprejme pozicijo stolpa, ki je bila generirana z metodo levo–desno. Nato program glede na trenutni stolpec, v kateri se stolp nahaja, shrani v seznam vsa mesta, ki predstavljajo ozemlje in prehodno pot. V naslednjem koraku s pomočjo seznama pridobi pozicijo prehodne poti, ki se v stolpcu nahaja najbližje našemu stolpu in s pomočjo te pozicije izračuna mesto, ki se nahaja zraven poti. To mesto predstavlja novo pozicijo stolpa. Na zadnje program preveri, če je na novo najdena pozicija že zasedena. V primeru, da je zasedena, program ponovno kliče naključno metodo in se vse skupaj ponovi. V nasprotnem primeru pa trenutno mutiranemu genu – stolpu nastavimo novo pozicijo. Ko se metoda levo–desno dokončno izvede, se vrne mutirani gen in s tem zaključi proces mutacije.

Postopek delovanja metode levo–desno lahko predstavimo z naslednjo psevdokodo:

ZAČETEK

Metoda kot parameter sprejme spremenjeni gen iz metode mutacija levo–desno.

PONAVLJAJ, DOKLER pozicija trenutnega kvadrata < dolžina ozemlja.

V seznam shrani mesta kvadratov v trenutnem stolpcu, kjer se nahaja prehodna pot (med iskanjem se premikamo po en kvadrat gor ali dol v stolpcu).

KONEC PONAVLJANJA

ČE seznam mest prehodnih poti ni prazen, **POTEM**

izračunaj pozicijo kvadratka poti, ki se nahaja najbližje trenutni poziciji.

Pridobi pozicijo kvadratka, ki se nahaja zraven poti.

ČE je pozicija zasedena, **POTEM**

ponovno kliči metodo naključna mutacija,

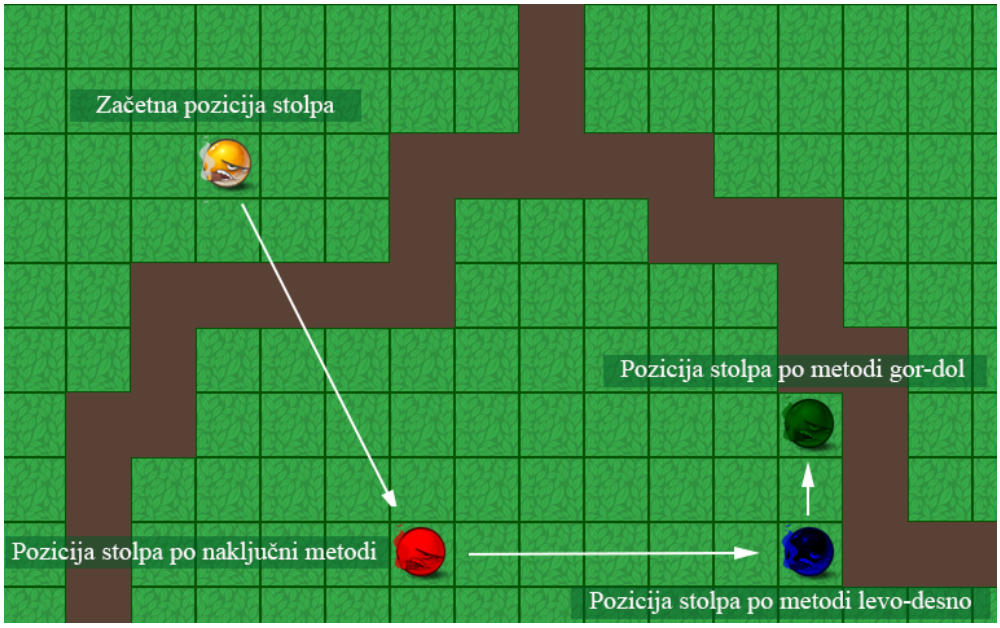
DRUGAČE

stolpu shrani novo pridobljeno pozicijo.

Vrni mutirani gen – stolp.

KONEC

Za lažje razumevanje je spodaj prikazano, kako poteka zamenjava pozicije pri metodi gor-dol (Slika 5.8).



Slika 5.8: Metoda gor–dol.

5.7 Selekcija

Prvi genetski operator, ki smo ga uporabili v algoritmu, je selekcija. Deluje tako, da v prvem koraku izbere najbolje ocenjeni postavitvi (elitizem), v ostalih korakih pa izbira naključno. S tem se dosega, da bo del potomcev vedno podedoval lastnosti najboljših postavitev prejšnjih generacij. To omogoča hitrejšo pot do dobre rešitve problema oziroma do dobre postavitve.

Tukaj se pojavi vprašanje, zakaj potem ne uporabimo za vse potomce najboljše postavitve prejšnje generacije, namesto da druge izbiramo naključno. Odgovor na to vprašanje je v monotonosti rešitve, saj bi se rezultati slej kot prej začeli ponavljati na točki, ki ni rešitev problema in iz katere bi vse poteze vodile na slabše.

Podrobnejši opis posameznih metod selekcij lahko najdemo v razdelku 3.4.1.

5.8 Zaustavitveni pogoj

Zaustavitveni pogoj je funkcija, ki zaustavi izvajanje algoritma na določeni točki. Poznamo več različnih zaustavitvenih pogojev, nekaj jih je naštetih v razdelku 3.5 tega dokumenta.

Naš algoritem vsebuje dva kriterija za zaustavljanje, in sicer:

- ročno zaustavljanje algoritma,
- zaustavljanje v primeru, da določena postavitev trikrat zaporedoma doseže oceno 10.

Ročno zaustavljanje deluje tako, da ko se uporabniku programa zdi, da je postavitev dovolj dobra za igro ali pa je s trenutno postavitvijo dobil idejo za dobro postavitev, ta stisne tipko Enter na tipkovnici in posledično se izvajanje algoritma zaustavi. Najboljša postavitev se shrani za kasnejšo uporabo. Možnost ročnega zaustavljanja je še posebej koristna zato, ker evolucijski algoritmi ne zagotavlja vedno najboljše rešitve, zato je dobro imeti možnost, da lahko mi sami zaustavimo algoritem, ko se nam rešitev zdi primerna.

Druga možnost je zaustavljanje v primeru, da določena postavitev trikrat zaporedoma doseže oceno 10. V primeru, da je ta pogoj izpolnjen, se algoritem neha izvajati in najboljša postavitev se shrani za kasnejšo uporabo. V nekaterih primerih se bo to zgodilo zelo hitro, v drugih primerih pa lahko traja tudi dlje časa, da program najde tako dobro postavitev. Vse to je odvisno predvsem od naključno generirane začetne populacije in sprememb med križanjem in mutacijo.

V naš algoritem bi lahko vključili tudi druge zaustavitvene kriterije, kot je kriterij zaustavljanja pri določeni generacije, ampak, ker ni zagotovila, da bo program našel dobro rešitev do določene generacije, smo se odločili zgolj za kriterija opisana zgoraj.

5.9 Začetni parametri

Za delovanje evolucijskega algoritma moramo najprej nastaviti vrednosti začetnih parametrov. Med te parametre štejemo:

- velikost populacije (označimo s pop_size ali V_{pop}),
- velikost kromosoma (število stolpov v eni postavitvi),
- verjetnost križanja (označimo s pc),
- verjetnost mutacije (označimo s pm).

Vsem začetnim parametrom smo spreminjali vrednosti skozi velika števila testiranj algoritma in smo obdržali tiste vrednosti, ki so se najbolje izkazale. Več o testiranjih je prikazano in opisano v razdelku 5.10.

5.9.1 Velikost populacije

Velikost populacije pomeni, koliko postavitev – kromosomov bo v populaciji oziroma v eni generaciji. Če gledamo na to iz stališča igre, v primeru, da nastavimo velikost populacije 100, bo program generiral v začetku izvajanja 100 različnih postavitev. Ker bi program s toliko postavitvami potreboval zelo veliko časa za iskanje dobre rešitve, smo mi v našem algoritmu nastavili parameter velikost populacije na vrednost 10. Ta parameter označujemo z oznako pop_size ali V_{pop}.

5.9.2 Velikost kromosoma

Nastavljena vrednost pove, koliko stolpov – genov bo v eni postavitvi – kromosomu. Manj kot je potrebnih stolpov za dobro rešitev, bolje je. Slednje zato, ker če gledamo iz stališča igralca, ki igra računalniško igro Obrani ozemlje brez algoritma, je njegov cilj imeti čim

boljšo postavitev s čim manjšim številom stolpov, saj posledično porabi manj denarja. V primeru, da nastavimo premalo število stolpov, se rešitev ne bo nikoli približala optimalni rešitvi, ker ne bodo zmožni uničiti vseh napadalnih enot.

V našem primeru je parameter velikosti postavitev nastavljen na 6, saj se je program s to vrednostjo najbolje izkazal v velikem številu testiranj na izbranih testnih ozemljih.

5.9.3 Verjetnost križanja

Verjetnost križanja nam pove, kakšen odstotek možnosti je, da bo prišlo do križanja med izbranima postavitvama. V našem primeru je verjetnost križanja nastavljena na 0.5, kar pomeni, da je 50 odstotkov možnosti, da se bo križanje izvedlo. Ta parameter označujemo z oznako p_c.

5.9.4 Verjetnost mutacije

V našem primeru je ta vrednost malo večja kot običajno, saj mutacija s svojimi dodatnimi metodami predstavlja zelo pomemben del našega algoritma in zelo pripomore k iskanju dobre rešitve v kratkem času. Nastavili smo vrednost 0.3, kar pomeni, da je 30 odstotkov možnosti, da se bo mutacija izvedla. Ta parameter označujemo z oznako p_m.

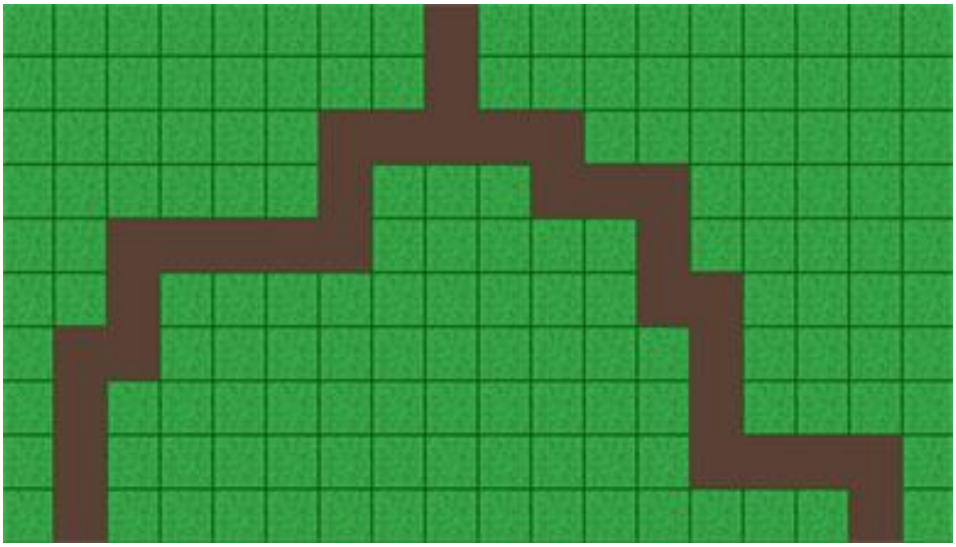
5.10 Testiranja in rezultati

Izdelani evolucijski algoritem smo testirali na tri načine, in sicer:

- testiranje iskanja rešitve z različnimi začetnimi parametri,
- testiranje vpliva dodatnih metod mutacije na iskanje rešitve,
- testiranje iskanja rešitve na različnih ozemljih.

Iskanje z evolucijskemu algoritmu poteka vsakič na drugačni populaciji, kar pomeni, da ne bo vedno našel rešitev v točno določeni generaciji. V nekaterih primerih bo našel rešitev zelo hitro, spet v drugih primerih pa bo potreboval več časa. To je predvsem odvisno od

naključno generirane začetne populacije in sprememb, ki jih prinašajo genetski operatorji. Da bi dobili približno predstavo, kako dolgo išče algoritem rešitev v določenih pogojih, smo za vsako vrednost parametra izvedli 30 krogov testiranja. Na podlagi teh lahko na koncu izračunamo povprečno število generacij, ki jih je algoritem potreboval za iskanje. Testiranja smo izvajali na ozemlju s prehodnimi potmi, ki se delijo na več poti (z izjemo pri testiranju različnih ozemelj). To ozemlje lahko vidimo spodaj na sliki (Slika 5.9). Zeleni kvadratki predstavljajo ozemlje, kjer lahko postavljamo stolpe, rjavi kvadratki pa predstavljajo prehodno pot. Izbira smeri pomika napadalne enote po teh poteh je naključna, zato se ocene ne pomikajo v vsakem koraku navzgor, kar je tudi opazno na grafih, ki so vključeni v nadaljevanju tega poglavja (podrobneje je to prikazano v razdelku 5.10.2).



Slika 5.9: Izbrano testno ozemlje.

5.10.1 Testiranje iskanja rešitve z različnimi začetnim parametri

Glavni cilj testiranja z različnimi parametri je ugotoviti, koliko generacij povprečno potrebuje algoritem, da najde rešitev z najboljšo oceno (v našem primeru najboljšo oceno opisuje število 100000, v skrajšani obliki 10). Posledično smo želeli spoznati, kako posamezni parametri vplivajo na hitrost in kvaliteto iskanja rešitve ter katere vrednosti parametrov dosegajo najboljše rezultate. Testirali smo algoritem s spremembo naslednjih začetnih parametrov:

- velikost kromosoma,

- verjetnost križanja,
- verjetnost mutacije.

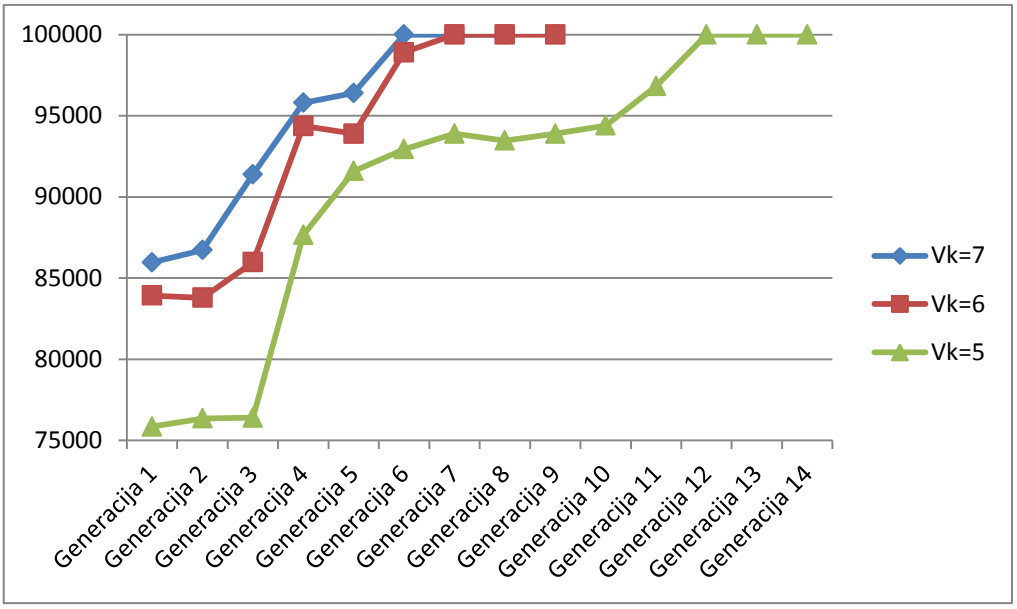
Prvi parameter, s katerim smo izvajali teste, je velikost kromosoma. Kot je že omenjeno v razdelku 5.9.2, nam ta parameter pove, koliko stolpov bo v eni postavitvi – kromosomu. Za testiranje smo izbrali vrednosti 5, 6 in 7. Tukaj je potrebno omeniti, da je za igralca pomembno, da uporabi čim manjše število stolpov, saj posledično porabi manj denarja za postavitve.

Tabela 5.1 prikazuje povzete rezultate 30 ponovitev in sicer prikazuje najboljše, najslabše in povprečne vrednosti. Iz teh rezultatov lahko razberemo, da je število generacij potrebnih za iskanje dobre rešitve (ocena 10) večje v primeru, ko uporabimo manjše število stolpov. Kar pomeni, da večja kot bo velikost kromosoma, prej bo algoritem našel dobro rešitev. Ker pa je naš cilj poiskati dobro rešitev s čim manjšim številom stolpov, smo se pri testiranju pomikali od večje vrednosti do manjše in tako skušali ugotoviti, katero je najmanjše število stolpov v naši igri, s katerimi lahko algoritem najde rešitev z oceno 10. Prišli smo do zaključka, da je najmanjša vrednost, pri kateri lahko postavitev uniči vse napadalne enote, 5 stolpov. Testirali smo tudi vrednost 4 in sicer v 1000 generacijah, vendar je ta dosegla najvišjo oceno 99916, kar pomeni, da ni nikoli uničila vseh napadalnih enot.

Tabela 5.1: Rezultati testiranj pri različnih velikostih kromosoma.

Velikost kromosoma (Vk)	**7**	**6**	**5**
Najboljši krog	6. generacija	7. generacija	12. generacija
Najslabši krog	20. generacija	25. generacija	70. generacija
Povprečno	9. generacija	14. generacija	31. generacija

Graf spodaj prikazuje izboljšave za posamezne velikost kromosoma v enem od krogov testiranja. Za prikaz smo izbrali najboljše kroge, kar pomeni, da je algoritem v teh krogih potreboval najmanjše število generacij za iskanje rešitev.



Naslednji parameter, s katerim smo izvajali teste, je verjetnost križanja. Kot je že omenjeno v razdelku 5.9.3, nam ta parameter pove, kakšen odstotek možnosti je, da bo prišlo do križanja med izbranima postavitvama. Za testiranje smo izbrali vrednosti 0.2 (20 %), 0.5 (50 %) in 0.8 (80 %). Ker je bil v tem primeru poudarek predvsem na verjetnosti križanja, smo za čas testiranja znižali verjetno mutacije in smo jo nastavili na 0.2 (20 %).

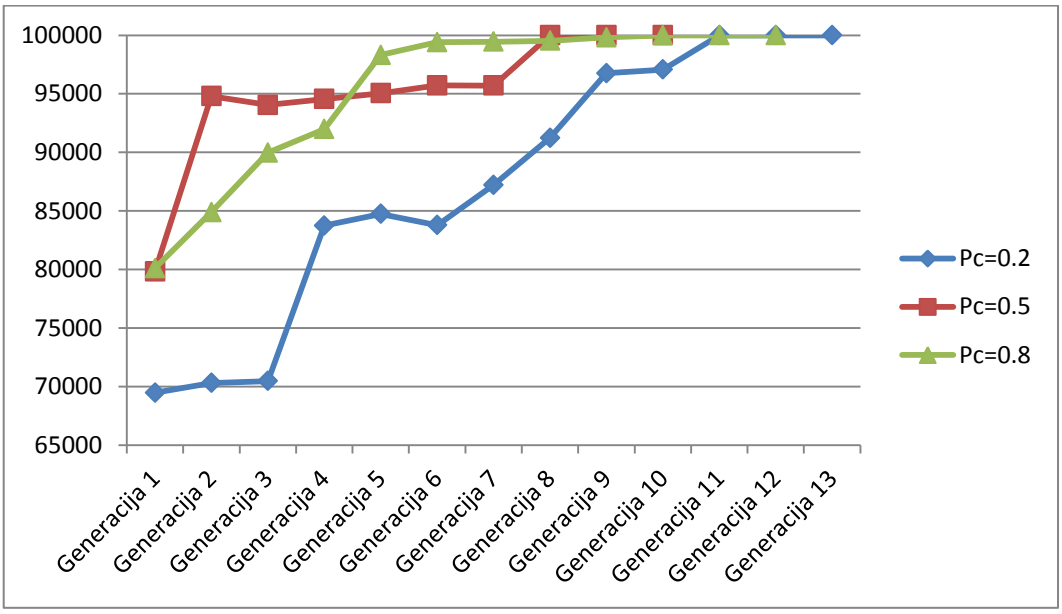
Tabela 5.2 prikazuje povzete rezultate 30 ponovitev in sicer prikazuje najboljše, najslabše in povprečne vrednosti. Iz teh rezultatov lahko razberemo, da ima verjetno križanja vpliv na hitrost iskanja rešitev, vendar v našem algoritmu ta vpliv ni zelo velik. Razlog za manjši vpliv tiči v enem od načinov izbiranja potomcev, elitizmu. Elitizem v naslednjo generacijo vedno prenese dve najboljši postavitvi, kar preprečuje, da bi se kvaliteta rešitve slabšala. To je posebej koristno v primeru, ko je verjetno križanja zelo visoka, saj se v tem primeru lahko zgodi, da preveliko število križanj poslabša že obstoječe dobre rešitve. Če opazujemo dano tabelo, lahko opazimo, da se je verjetno 0.2 (20 %) izkazala kot najpočasnejša pri iskanju rešitve. Kot najhitrejša pa se je izkazala vmesna vrednost in sicer vrednost 0.5 (50 %), vendar vrednost 0.8 (80 %) ni veliko zaostajala za njo.

Med testiranjem smo lahko prav tako opazili, da se pri višji nastavljeni vrednosti rešitve pogosteje ponavljajo. Ta problem bi lahko rešili z nastavljeno večjo vrednostjo verjetnosti mutacije, ki bi v rešitve dodajala element naključnosti.

Tabela 5.2: Rezultati testiranj pri različnih vrednostih verjetnosti križanja.

Verjetnost križanja (Pc)	0.2 (20%)	0.5 (50%)	0.8 (80%)
Najboljši krog	11. generacija	8. generacija	10. generacija
Najslabši krog	44. generacija	27. generacija	27. generacija
Povprečno	20. generacija	16. generacija	17. generacija

Graf spodaj prikazuje izboljšave za posamezne vrednost verjetnosti križanja v enem od krogov testiranja, ki jih prikazuje Tabela 5.2. Za prikaz smo izbrali najboljše kroge, kar pomeni, da je algoritem v teh krogih potreboval najmanjše število generacij za iskanje rešitev.



Zadnji parameter, s katerim smo izvajali teste, je verjetnost mutacije. Kot je že omenjeno v razdelku 5.9.4, nam ta parameter pove, kakšen odstotek možnosti je, da bo prišlo do mutacije v izbranem kromosomu - postavitvi. Za testiranje smo izbrali vrednosti 0.1 (10 %), 0.5 (50 %) in 0.9 (90 %). Verjetnost križanja smo nastavili na vrednost, ki se je v prejšnjem testiranju izkazala kot najboljša, torej 0.5 (50 %).

Tabela 5.3 prikazuje povzete rezultate 30 ponovitev, in sicer prikazuje najboljše, najslabše in povprečne vrednosti. Iz teh rezultatov lahko razberemo, da ima v našem algoritmu verjetnost mutacije velik vpliv na hitrost iskanja rešitve. Če opazujemo tabelo, lahko opazimo, da se je najbolje izkazala vrednost 0.9 (90 %), najslabše pa se je izkazala

vrednost 0.1 (10 %). Razlog za takšne rezultate tiči v ohranjanju dobrih rešitev s pomočjo elitizma in v dodatnih metodah mutacije, ki smo jih izdelali, saj te metode postavljajo stolpe na pozicije ob poti, kjer imajo največji vpliv na napadalne enote in posledično ne morajo zelo poslabšati rešitve, ko se mutacija izvede. Vendar ti rezultati niso povsem realni, kar bi bilo bolje opazno v primeru, če bi uporabili zgolj naključno metodo mutacije in ne bi uporabljali elitizma, saj če bi verjetnost mutacije preveč povečali, bi se algoritem izrodil v zgolj naključno iskanje dobre rešitve. V primeru če pa bi bila verjetnost mutacije premajhna, pa bi prišlo do problema, da se izgubljen genetski material zaradi križanja ne bi obnovil. To trditev lahko dokazujemo z rezultati, ki jih kaže Tabela 5.4.

Glede na zgornje ugotovitve in običajno uporabljene vrednosti v praksi smo prišli do zaključka, da bi v našem algoritmu bilo najbolje uporabljati verjetnost mutacije okoli 0.3 (30 %), saj bi večja vrednost verjetnosti mutacije pomenila zgolj naključno iskanje dobre rešitve in tega postopka ne bi več mogli imenovati evolucijski algoritem.

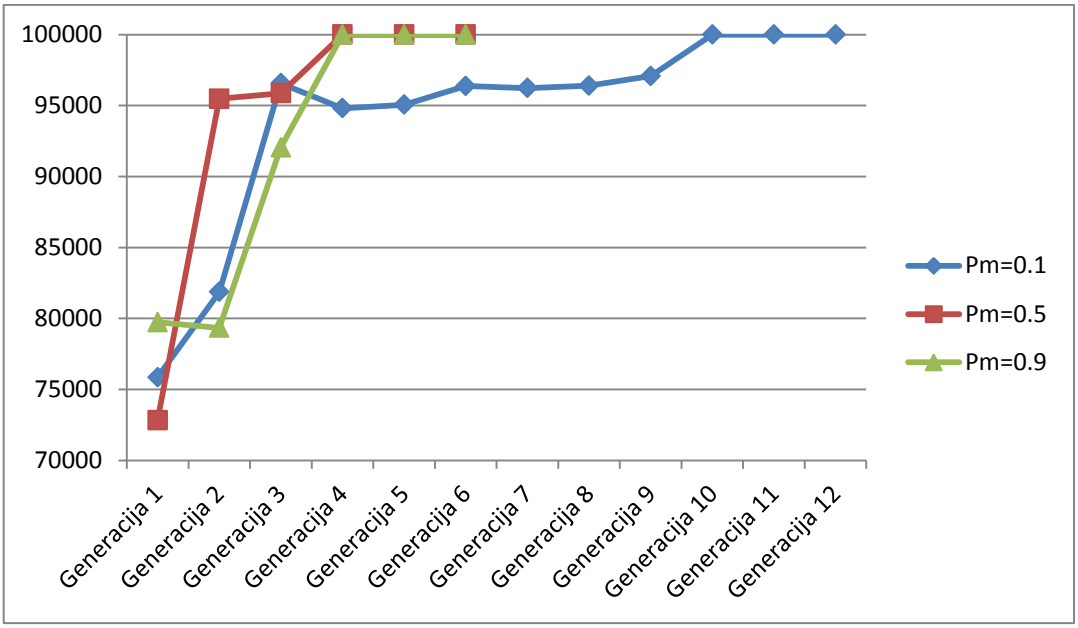
Tabela 5.3: Rezultati testiranj pri različnih vrednostih verjetnosti mutacije.

Verjetnost mutacije (Pm)	**0.1 (10 %)**	**0.5 (50 %)**	**0.9 (90 %)**
Najboljši krog	10. generacija	4. generacija	4. generacija
Najslabši krog	50. generacija	19. generacija	9. generacija
Povprečno	22. generacija	11. generacija	9. generacija

Tabela 5.4: Rezultati testiranja le z naključno metodo mutacije in brez elitizma.

Verjetnost mutacije (Pm)	**0.1 (10 %)**	**0.9 (90 %)**
Povprečno	preko 2000 generacij	700. generacija

Graf spodaj prikazuje izboljšave za posamezne vrednost verjetnosti mutacije v enem od krogov testiranja, ki jih prikazuje Tabela 5.3. Za prikaz smo izbrali najboljše kroge, kar pomeni, da je algoritem v teh krogih potreboval najmanjše število generacij za iskanje rešitev.



5.10.1 Testiranje vpliva dodatnih metod mutacije na iskanje rešitve

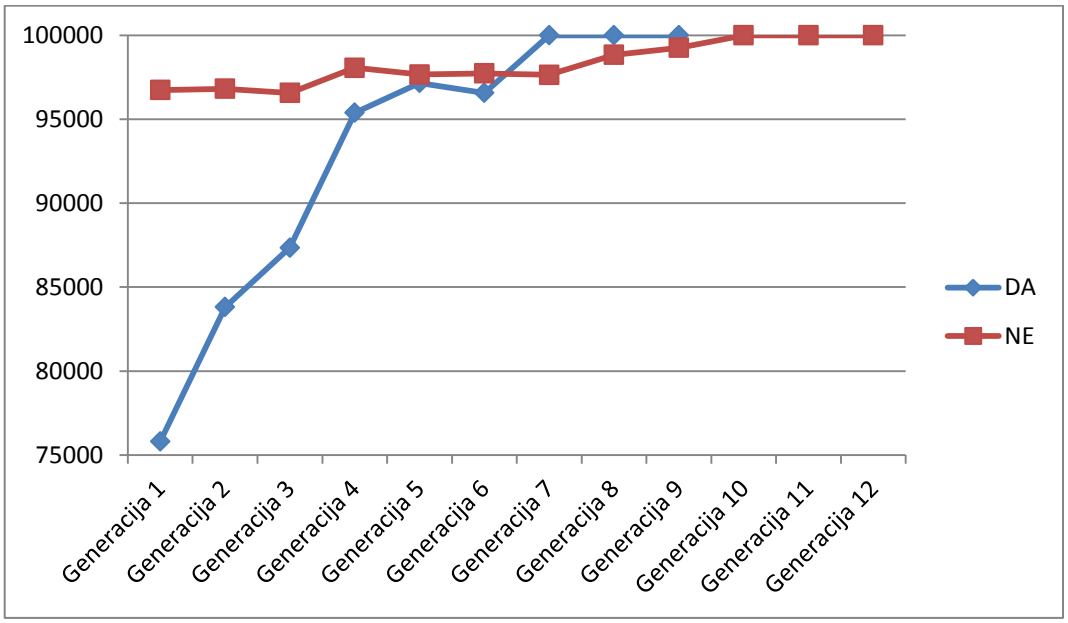
Glavni cilj testiranja vpliva dodatnih metod mutacije (opisanih v razdelku 5.6) je ugotoviti, kako te metode pripomorejo k iskanju rešitve in kako hitro bi iskanje potekalo brez njih. Hitrost iskanja rešitve smo vrednotili glede na to, koliko generacij je povprečno potreboval algoritem, da je našel rešitev z najboljšo oceno (v našem primeru najboljšo oceno opisuje število 100000, v skrajšani obliki 10).

Tabela 5.5 prikazuje povzete rezultate 30 ponovitev, in sicer prikazuje najboljše, najslabše in povprečne vrednosti. Iz teh rezultatov lahko razberemo, da dodatne metode zelo pripomorejo k hitrosti iskanja rešitve. Prav tako v primeru, ko testiramo z večjim številom stolpov, kot pa je to potrebno za uničenje vseh napadalnih enot, lahko pričakujemo boljšo rešitev z dodatnimi metodami mutacije, saj bodo na koncu vsi stolpi postavljeni ob poteh, kjer imajo največji vpliv. Glede na povprečne vrednosti je algoritem z dodatnimi metodami poiskal rešitev v trikrat krajšem času.

Tabela 5.5: Rezultati testiranja vpliva dodatnih metod mutacije.

Dodatne metode	DA	NE
Najboljši krog	7. generacija	10. generacija
Najslabši krog	23. generacija	118. generacija
Povprečno	15. generacija	44. generacija

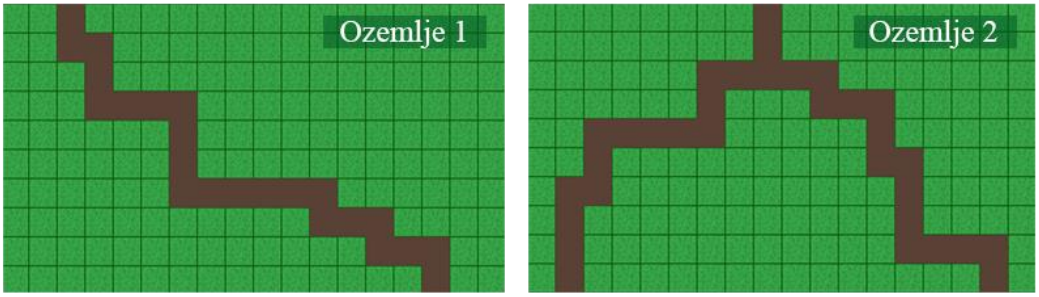
Graf spodaj prikazuje izboljšave za testiranje brez dodatnih metod in testiranje z dodatnimi metodami mutacije v enem od krogov testiranja, ki jih prikazuje tTabela 5.5. Za prikaz smo izbrali najboljše kroge, kar pomeni, da je algoritem v teh krogih potreboval najmanjše število generacij za iskanje rešitev. Na grafu lahko opazimo, da je najboljši krog pri testiranju brez dodatnih metod že imel zelo dobro generirano začetno populacijo, saj je že v prvi generaciji dosegel oceno 9. Iz tega razloga je potreboval samo 10 generacij za iskanje dobre rešitve. Če bi bila začetna populacija slabše generirana, bi algoritem potreboval precej več časa za iskanje dobre rešitve, kar tudi dokazujejo povprečne vrednosti, prikazane v tabeli zgoraj.



5.10.2 Testiranje iskanja rešitve na različnih ozemljih

Glavni cilj testiranja različnih ozemelj je ugotoviti, kako ta vplivajo na hitrost iskanja rešitve. Hitrost iskanja rešitve smo vrednotili glede na to, koliko generacij je povprečno

potreboval algoritem, da je našel rešitev z najboljšo oceno (v našem primeru najboljšo oceno opisuje število 100000, v skrajšani obliki 10). Testiranje smo izvedli na dveh ozemljih, in sicer na ozemlju z eno potjo in ozemlju, kjer se pot deli na dve poti. Izbrana ozemlja so prikazana spodaj (Slika 5.10).



Slika 5.10: Testna ozemlja.

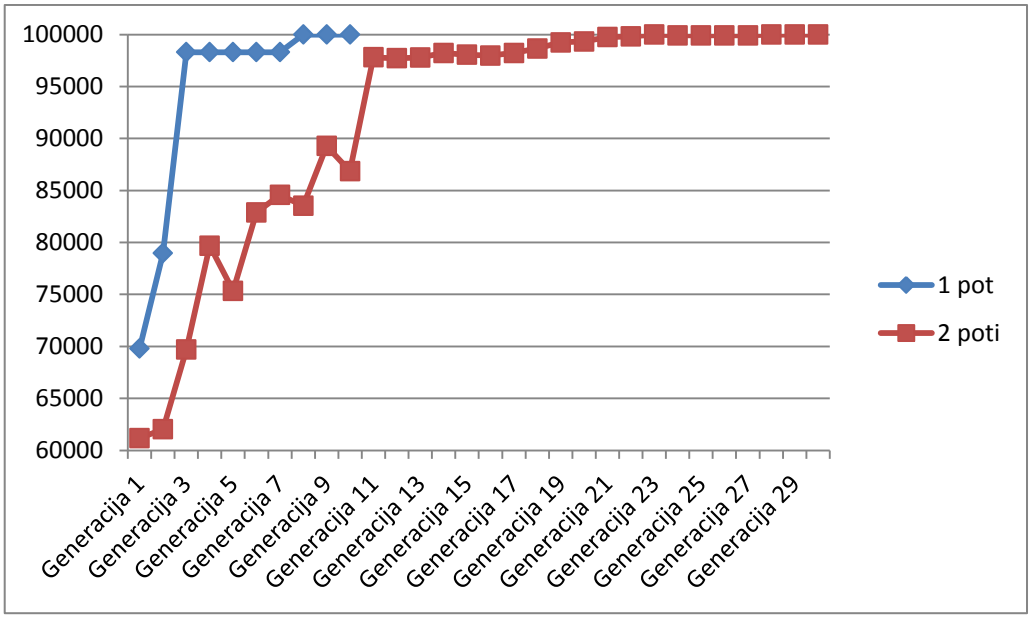
Tabela 5.6 prikazuje povzete rezultate 30 ponovitev in sicer prikazuje najboljše, najslabše in povprečne vrednosti. Iz teh rezultatov lahko razberemo, da število poti oziroma delitev poti na ozemlju vpliva na hitrost iskanja rešitve. Glede na rezultate v tabeli, lahko opazimo, da več kot je poti na ozemlju, dlje časa bo algoritem potreboval za iskanje rešitve. Razlog je v tem, da za več poti potrebujemo večje število stolpov oziroma dobro postavitev stolpov, ki so na voljo, če želimo uničiti vse napadalne enote. Nasprotno je v primeru, ko imamo manj poti, saj potrebujemo manj stolpov ali pa je dovolj, če je pravilno postavljenih le delež stolpov, ki so na voljo. Če uporabimo enako število stolpov za ozemlje z eno potjo in ozemlje z dvema potema, lahko opazimo, da bi pri ozemlju z eno potjo zadostovalo manjše število stolpov.

Tabela 5.6: Rezultati testiranja algoritma na različnih ozemljih.

Ozemlje	**Enostavno (1 pot)**	**Zahtevnejšo (2 poti)**
Najboljši krog	2. generacija	5. generacija
Najslabši krog	8. generacija	28. generacija
Povprečno	4. generacija	15. generacija

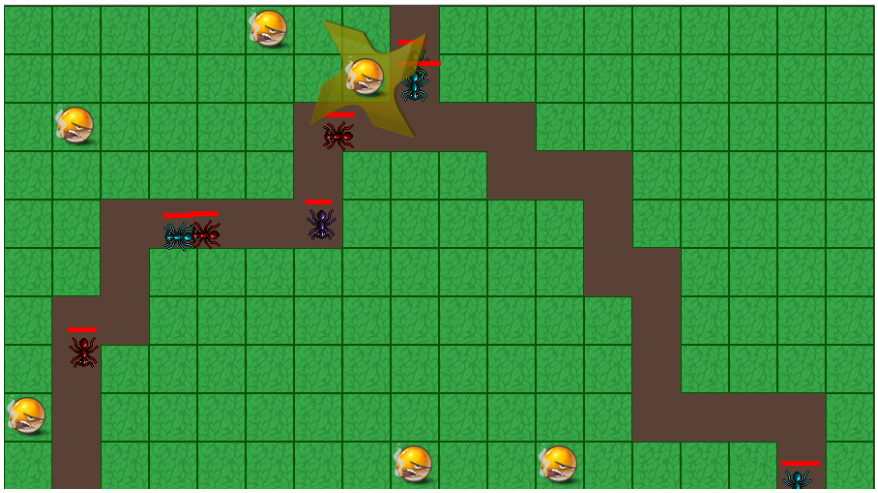
Graf spodaj prikazuje izboljšave za testiranje brez dodatnih metod in testiranje z dodatnimi metodami mutacije v enem od krogov testiranja, ki jih prikazuje Tabela 5.6. Za prikaz smo tokrat izbrali najslabše kroge, kar pomeni, da je algoritem v teh krogih potreboval največje

število generacij za iskanje rešitev. Najslabše kroge smo izbrali zato, da natančneje prikažemo, kako poteka izboljševanje rešitev pri zahtevnejših poteh, kjer so smeri napadalnih enot naključne. Posledično posamezne postavitve ne dobijo vedno tako dobrih ocen, kot so jih dobile v prejšnji generaciji, kar lahko tudi opazimo na grafu.

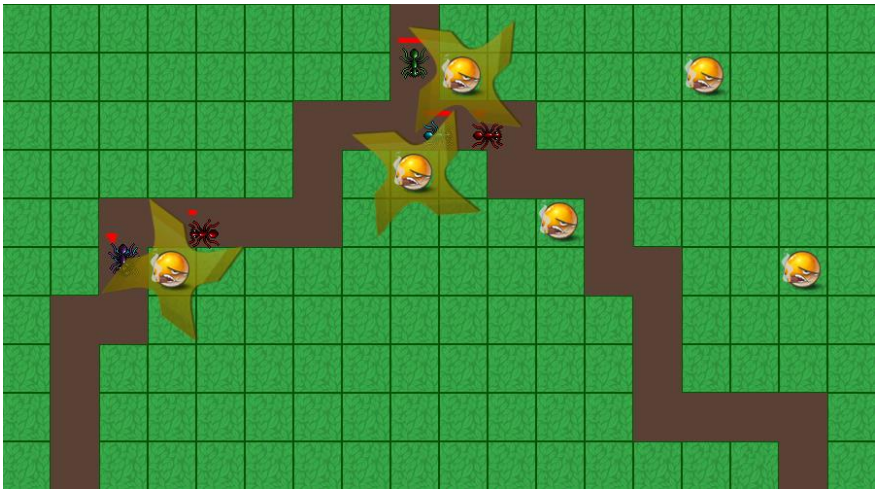


5.10.3 Primer iskanja dobre rešitve

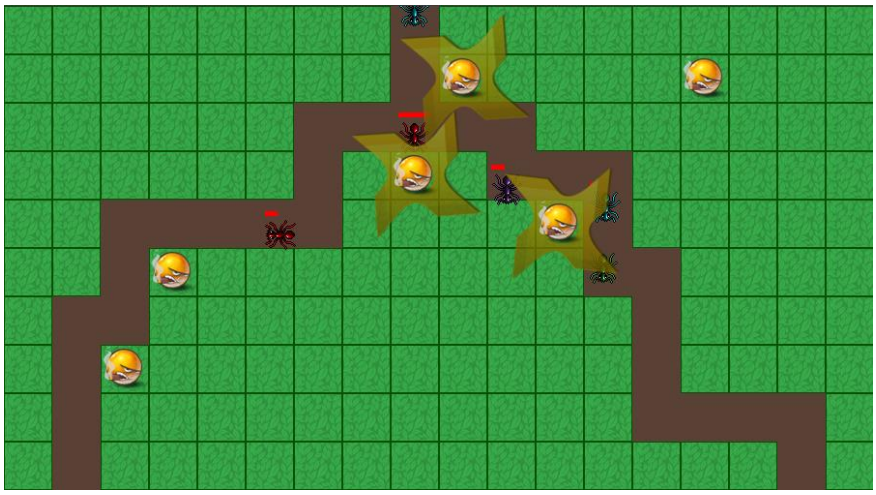
Na slikah spodaj je prikazan potek iskanja dobre rešitve. Na prvi sliki je prikazana ena od začetnih naključno generiranih postavitev (Slika 5.11). Naslednji dve sliki prikazujeta vmesni napredek iskanja rešitve v 5. in 10. generaciji (Slika 5.12 in Slika 5.13). Zadnja slika pa prikazuje najdeno dobro rešitve v 14. generaciji, kar pomeni, da je postavitev, ki je na prikazana na sliki, uspela uničiti vse napadalne enote (Slika 5.14).



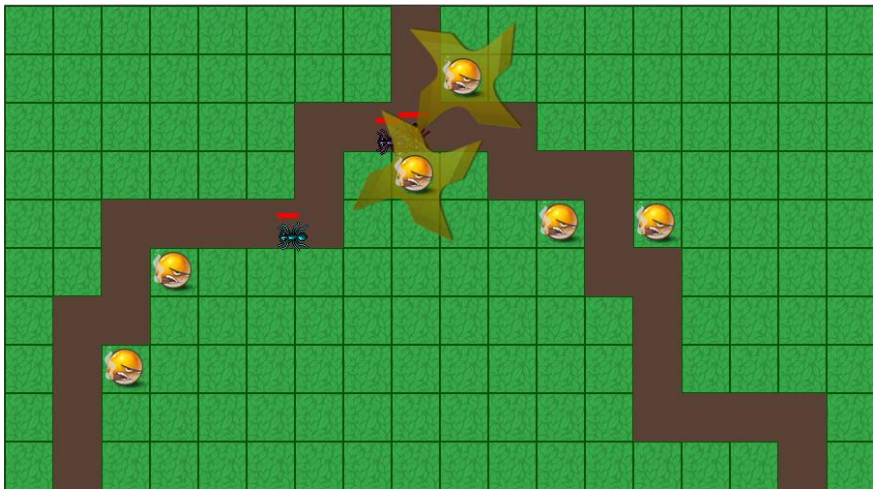
Slika 5.11: Začetna postavitev – Generacija 1.



Slika 5.12: Vmesna postavitev - Generacija 5.



Slika 5.13: Vmesna postavitev - Generacija 10.



Slika 5.14: Najdena dobra rešitev – Generacija 14.

6 SKLEP

V okviru diplomskega dela smo razvili evolucijski algoritem za iskanje dobre postavitve obrambnih stolpov v računalniški igri Obrani ozemlje. S pomočjo algoritma smo nato ustvarili inteligentnega pomočnika, ki na podlagi najdene dobre rešitve igralcu prikaže namige za postavitev stolpov. Evolucijski algoritmi veljajo za zelo robustne optimizacijske algoritme, ki se uporabljajo za reševanje zahtevnejših problemov, kjer običajne metode odpovejo.

Naš algoritem smo testirali v treh kategorijah. Testi so pokazali, da algoritem najde rešitev v sprejemljivem času. Poleg tega pa smo s pomočjo podatkov, ki smo jih pridobili med testiranjem, algoritem še izboljšali in nastavili bolj učinkovite vrednosti začetnih parametrov. Prav tako smo ugotovili, da naš algoritem z dodatnimi metodami mutacije zelo hitro najde dobro rešitev v primerjavi z zgolj naključno mutacijo. S tem smo dosegli cilje, ki smo si jih zastavili v uvodu.

Algoritem bi lahko izpopolnili na več načinov. Lahko bi uporabili druge načine križanja, kot je recimo dvotočkovno križanje, prav tako pa bi lahko metode mutacije uporabljali izmenično, saj bi to pripomoglo še k večji raznolikosti rešitev. Zelo koristno bi bilo samo izvajanje aplikacije in algoritma pospešiti, saj je trenutno izvajanje precej počasno in zamudno, kar je za igralca zelo nepraktično.

Največ težav med izdelovanjem diplomske naloge nam je povzročalo nastavljanje začetnih parametrov, saj se je algoritem v nekaterih primerih bolje izkazal z vrednostmi, ki niso običajne v praksi. Vzrok za to je bil v dodatno izdelanih metodah mutacije, zato smo vrednosti nastavili bližje tistim, uporabljenim v praksi, saj v nasprotnem primeru ne bi več šlo za evolucijski algoritem, ker bi iskanje postalo zgolj naključno.

V prihodnosti bi želeli igro in algoritem še izboljšati. To bi storili tako, da bi izvajanje in iskanje še bolj pospešili in sistem za namige med igranjem nadgradil z velikim številom težavnostnih nivojev.

7 VIRI, LITERATURA

[1] *Algoritem, Wikipedia*. 2013. Dostopno na: http://sl.wikipedia.org/wiki/Algoritem [10. 6 2013].

[2] Banzhaf, W., Nordin, P., Keller, R., Francone, D. *Genetic Programing - An Introduction*. San Francsisco: Morgan Kaufmann, 1998.

[3] Cormen, T., Leiserson, C., Riverst, R., Stein, C. *Introduction to algorithms*. Cambridge: Massachusetts Institute of Technology, 2001.

[4] Dovgan, E. *Evolucijski algoritem za optimiranje prevoza tovora med dvema lokacijama s skupino vozil: diplomska naloga na univerzitetnem študiju*. Ljubljana: Fakulteta za računalništvo in informatiko, 2008.

[5] Eiben, A., Smith, J. *Introduction to Evolutionary Computing*. Berlin: Springer, 2003.

[6] Guid, N., Strnad, D. *Umetna inteligenca*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko, 2007.

[7] Janssen, C. *XNA Game Studio, Techopedia*. 2013. Dostopno na: http://www.techopedia.com/definition/16463/xna-game-studio [12. 6 2013].

[8] Kokol, P., Hleb Babič, Š., Podgorelec, V., Zorman, M. *Inteligentni sistemi*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko, 2000.

[9] Mernik, M., Črepinšek, M., Žumer, V. *Evolucijski algoritmi*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko, 2003.

[10] *Računalniška igra, Wikipedia*. 2013. Dostopno na: http://sl.wikipedia.org/wiki/Ra%C4%8Dunalni%C5%A1ka_igra [10. 6 2013].

[11] *Tower defense, Wikipedia*. 2013. Dostopno na: http://en.wikipedia.org/wiki/Tower_defense [10. 6 2013].

[12] *Videoigra, Wikipedia*. 2013. Dostopno na: http://sl.wikipedia.org/wiki/Videoigra [10. 6 2013].

[13] *XNA Game Studio 4.0 Refresh, MSDN*. 2013. Dostopno na: http://msdn.microsoft.com/en-us/library/bb200104.aspx [12. 6 2013].

[14] Yu, X., Gen, M. *Introduction to Evolutionary Algorithms*. London: Springer, 2010.

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

IZJAVA O USTREZNOSTI DIPLOMSKEGA DELA

Spodaj podpisani/-a MATEJ ČREPINŠEK izjavljam, da je
(ime in priimek mentorja/-ice)

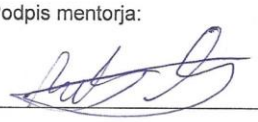
študent TILEN HOSNAR izdelal diplomsko
(ime in priimek študenta/-ke)

delo z naslovom: UPORABA EVOLUCIJSKIH ALGORITMOV ZA POSTAVITEV STOLPOV V RAČUNALNIŠKI IGRI OBRANI OZEMLJE
(naslov diplomskega dela)

v skladu z odobreno temo diplomskega dela, Navodili za pisanje diplomskih del na dodiplomskih študijskih programih UM FERI in mojimi navodili.

Kraj in datum: 12. 9. 2013

Podpis mentorja:



Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

IZJAVA O AVTORSTVU

Spodaj podpisani/-a TILEN HOSNAR

z vpisno številko E1041143

sem avtor/-ica diplomskega dela z naslovom: UPORABA EVOLUCIJSKIH ALGORITMOV ZA POSTAVITEV STOLPOV V RAČUNALNIŠKI IGRI OBRANI OZEMLJE

(naslov diplomskega dela)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

doc. dr. MATEJ ČREPINŠEK

in somentorstvom (naziv, ime in priimek)

/

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela.
- soglašam z javno objavo elektronske oblike diplomskega dela v DKUM.

V Mariboru, dne 13.9.2013

Podpis avtorja/-ice:

Tilen Hosnar

Univerza v Mariboru

Fakulteta za elektrotehniko, računalništvo in informatiko

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE ZAKLJUČNEGA DELA IN OBJAVI OSEBNIH PODATKOV DIPLOMANTOV

Ime in priimek avtorja-ice: TILEN HOSNAR

Vpisna številka: E1041143

Študijski program: RAČUNALNIŠTVO IN INFORMACIJSKE TEHNOLOGIJE

Naslov zaključnega dela: UPORABA EVOLUCIJSKIH ALGORITMOV ZA POSTAVITEV STOLPOV V RAČUNALNIŠKI IGRI OBRAMBE OZEMLJE

Mentor: dr. MATEJ ČREPINŠEK

Somentor: /

Podpisani-a TILEN HOSNAR izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru. Zaključno delo sem izdelal-a sam-a ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija zaključnega dela je istovetna z elektronsko verzijo elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru.

Zaključno delo zaradi zagotavljanja konkurenčne prednosti, varstva industrijske lastnine ali tajnosti podatkov naročnika: _____ ne sme biti javno dostopno do _____(datum odloga javne objave ne sme biti daljši kot 3 leta od zagovora dela).

Podpisani izjavljam, da dovoljujem objavo osebnih podatkov, vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum zaključka študija, naslov zaključnega dela), na spletnih straneh in v publikacijah UM.

Datum in kraj: 12.9.2013, MARIBOR Podpis avtorja-ice: Tilen Hosnar

Podpis mentorja: _____
(samo v primeru, če delo ne sme biti javno dostopno)

Podpis odgovorne osebe naročnika in žig:
(samo v primeru, če delo ne sme biti javno dostopno) _____